Karlsruhe Institute of Technology

Institute of Experimental Particle Physics

Development of a C++/Python Interface for a New Track Finding Algorithm in Belle II

Entwicklung einer C++/Python-Schnittstelle für einen neuen Spurfindungsalgorithmus in Belle II

Bachelor's Thesis of
Bachelorarbeit von

Yannis Klügl

at the Department of Physics
an der Fakultät für Physik

Institute of Experimental Particle Physics
Institut für Experimentelle Teilchenphysik

Advisor: Prof. Dr. Torben Ferber
Referent

Coadvisor: Dr. Giacomo De Pietro
Korrefferent

14. November 2023 - 30. March 2024
14. November 2023 - 30. März 2024

ETP-Bachelor-KA/2024-01

Abstract

Over the last years, there have been many advancements in the area of artificial intelligence (AI) and neural networks, which also make their way into science applications. Researchers at the Institute of Experimental Particle Physics at the Karlsruhe Institute of Technology are exploring new ways to solve problems in particle physics using these achievements in AI research. One of those problems is the track finding for particles at the Belle II experiment in Tsukuba, Japan. Here, particles generated from the collision of an electron and a positron beam pass through a tracking detector, the central drift chamber (CDC), and leave a trail of hits. Analyzing these tracks allows to obtain information about the particle, for example, its momentum and charge.

As a classic approach, track finding algorithms based on the concepts of Legendre transformation and cellular automata are used. In a novel approach, a neural network that takes the hits in the detector as an input and produces reconstructed tracks as an output is now employed. This neural network, which is currently in development, is known as the CDC AI Track Finder (CAT Finder). The goal is to implement this tracking algorithm as a module in the C++ analysis software framework of the Belle II experiment (basf2) [1]. Contrary to most of basf2, the CAT Finder module is written in Python, which allows for quick prototyping and better readability but comes with the disadvantage of less performance than C++. This makes it necessary to efficiently interface between the two languages.

This thesis focuses on developing such an interface. Besides implementing the C++/Python interface ways to optimize it are also discussed, aiming to reduce the runtime and memory consumption of the CAT Finder module. To verify that the C++/Python interface, produces the same output as the CAT Finder, the track finding efficiency, fake rate, and resolution of transverse momentum are studied. These metrics are measured for multiple event types and beam-background scenarios. Lastly, the same benchmarks are performed on the established Legendre tracking algorithm and compared to the results of the CAT Finder and C++/Python interface.

Preface

This thesis builds upon the work of Lea Reuter [2], who provided the CAT Finder Python code and parts of a module to measure the tracking performance.
The event simulation uses the analysis software of the Belle II experiment [1]. CAT Finder and the C++/Python interface developed in this thesis aim to expand this software framework.
The beam-background data as well as a module for basf2 statistics was provided by Dr. Giacomo De Pietro.
The beam-backgrounds are centrally produced by the Belle II collaboration using MC simulations and can be found at
`/group/belle2/dataprod/BGOverlay/early_phase3/release-06-00-05/overlay/BGx1`
and
`/group/belle2/dataprod/BGOverlay/nominal_phase3/release-06-00-05/overlay/BGx1`
on KEKCC for low and high beam-background respectively. See also [3].

My contributions to the project are the implementation and optimization of the C++ code for the CAT Finder. Additionally, I developed benchmarks to quantify the performance of the C++/Python interface and other tracking algorithms. These are inspired by the benchmarks used by Lea Reuter for the CAT Finder.

Dr. Giacomo De Pietro and Prof. Torben Ferber proposed the studies conducted in this thesis.

Contents

Chapter 1

Introduction

1.1 The Belle II experiment

The Belle II detector is a particle detector located at the High Energy Accelerator Research Organization (高エネルギー加速器研究機構), also known as KEK, in Tsukuba, about 60 km north-east of Tokyo, Japan. The primary goals of Belle II are the search for new physics and more precise measurements of Standard Model parameters [4].
The detector is operated at the SuperKEKB accelerator complex, which is an asymmetric, circular e^+e^- accelerator using 4 GeV positrons and 7 GeV electrons [5]. It is designed to reach an instantaneous luminosity of 6.5×10^{35} cm^{-2} s^{-1}, about 30 times as high as its predecessor KEKB [6] [7]. The data is mainly collected at the $\Upsilon(4S)$ resonance at approximately 10.58 GeV, allowing for the production of B-meson pairs [4] [8].
The Belle II detector was first proposed in 2004 as an upgrade of the Belle experiment and data-taking began in 2018 [4] [9].

Like most modern particle detectors, the Belle II detector consists of several specialized detectors that are arranged in layers around the interaction region. From the center outwards these are three tracking detectors (PXD, SVD, CDC), two detectors for particle identification (TOP, ARICH), the electromagnetic calorimeter (ECL), and finally the K_L and muon detector (KLM). The arrangement of these detectors and other components is shown in Figure 1.1.
The following sections will discuss the Belle II tracking system in more detail.

1.2 Track finding at Belle II

1.2.1 Hardware

The track finding system at Belle II consists of three detectors: the pixel detector (PXD), the silicon vertex detector (SVD), and the central drift chamber (CDC). As the tracking algorithm studied in this thesis exclusively works with the CDC, the PXD and SVD are discussed only briefly.

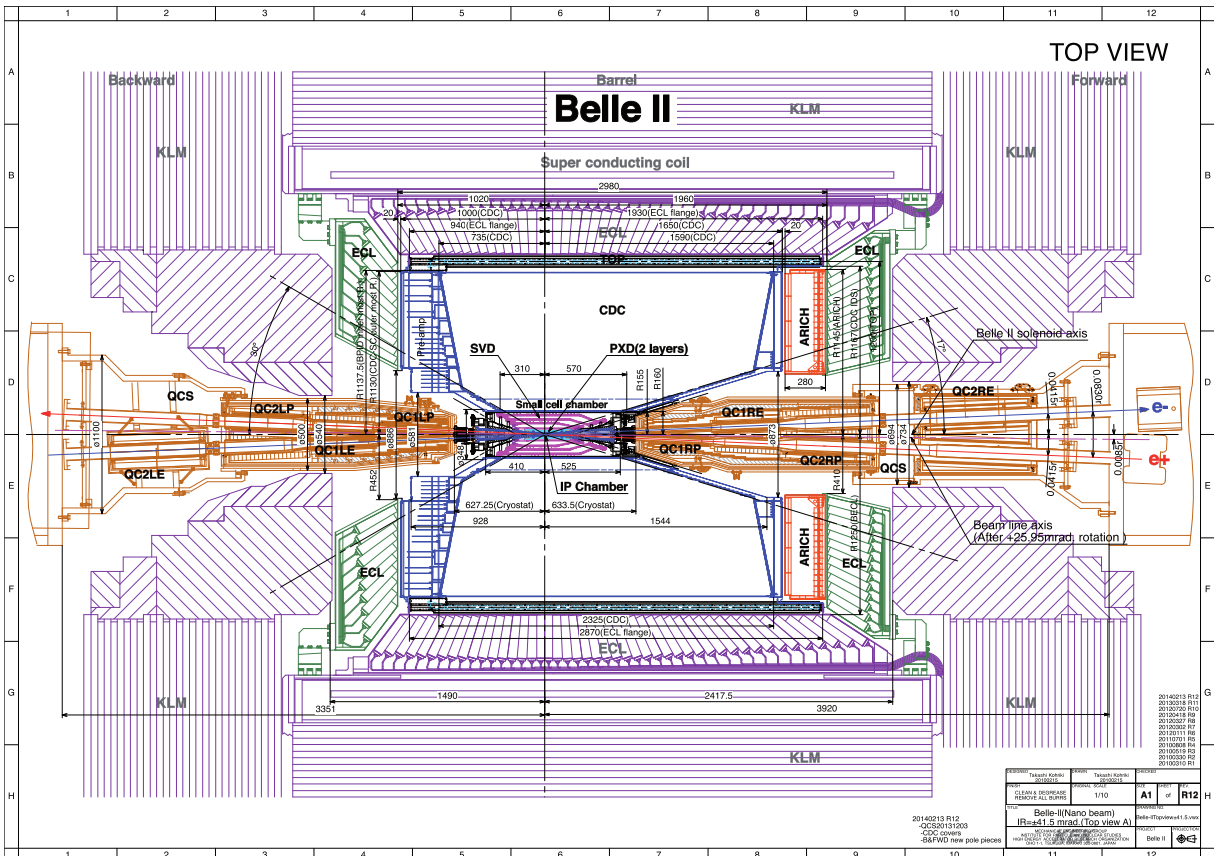


Figure 1.1: Top view of the Belle II detector showing the layer-like arrangement of the individual detectors. The beamline (orange) enters from the left and right. Image taken from [4].

The PXD consists of two layers of eight and twelve ladders based on depleted p-channel field effect transistor (DEPFET) technology. The ladders are created by combining two DEPFET modules for a total of 40 sensors with a resolution of 250 by 768 pixels each.
The SVD is built similarly with four layers of double-sided silicon strip detectors on 35 ladders and a total of 172 sensors. The sensors have a resolution of 768 by 768 strips per sensor on the first layer and 768 by 512 strips per sensor on the remaining layers [5].

The CDC is a cylindric chamber 226 cm in diameter. To make space for the SVD, PXD, and beamline, the center is spared along the z-axis with a minimum diameter of 32 cm [9]. This volume is filled by about 50 000 sense and field wires which are combined into 56 layers and 9 superlayers. The superlayers contain 6 layers each, except for the innermost superlayer which contains 8 layers. This and every second layer following is an axial layer where the wires are aligned parallel to the z-axis. In the remaining layers, the wires are skewed by 45.4 mrad to 74 mrad to form a stereo layer [5]. The direction of the skew alternates between the stereo layers. This skew allows information about the z-position of a particle to be reconstructed.
Figure 1.2 shows a quadrant of the cross-section of the CDC volume on the left side. The dark and light points represent the axial and stereo layers respectively. To the right, the alignment of the axial (top) and stereo (bottom) wires are shown. The skew of the stereo wires is exaggerated for illustration.

The CDC is filled with a mixture of 50 % helium and 50 % ethane. The performance of this gas mixture has been studied in the Belle experiment [9].

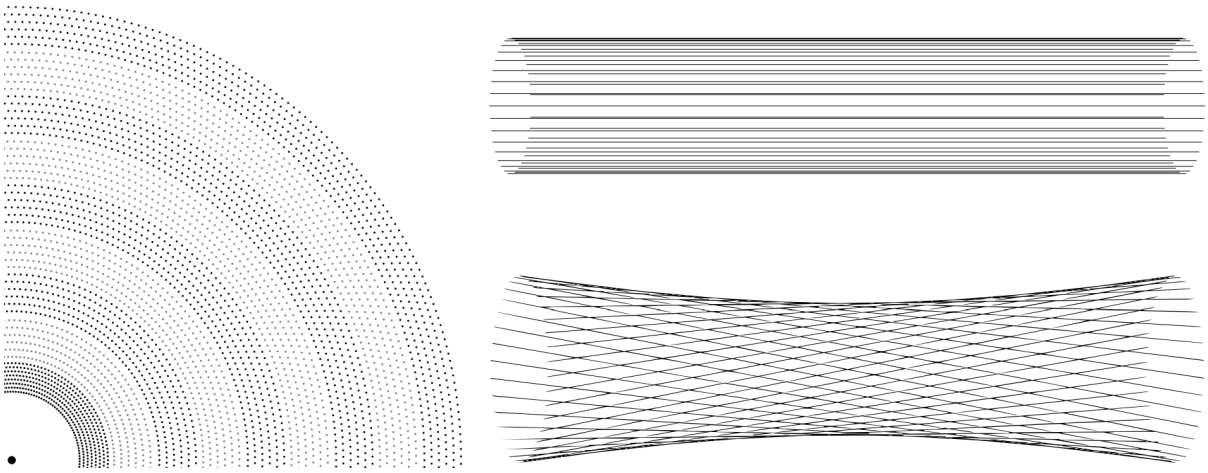


Figure 1.2: Wire configuration of the CDC. The left side shows a quadrant of the cross-section of the CDC, the right shows the wires as seen perpendicular to the z-axis with an axial layer on the top and a stereo layer on the bottom. Image taken from [5].

The CDC measurements strongly depend on several factors. It is trivial to see that the number of hits increases along with the beam-background. Beam-background occurs from losses in the particle beam that produce secondary particles in the detector [3]. The event type determines the number of tracks and thus also influences the number of hits that are generated. For B decays there are an average of 11 tracks per event while $e^+e^- \rightarrow \mu^-\mu^+$ events typically only generate 2 tracks and $e^+e^- \rightarrow \mu^+\mu^-$ events generate 2 to 6 tracks. For momenta below 300 MeV [5] the particle trajectory can be curved so strongly that it will return to the CDC or is unable to leave it. The tracks of these particles, called curlers, contain more CDC hits due to their longer trajectory in the CDC and are more challenging to reconstruct. An example of the CDC measurement of a single event can be seen in Figure 1.3.

1.2.2 Software

There are currently two established algorithms used for track finding in Belle II, a global algorithm using Legendre transformations and a local algorithm based on a cellular automaton. The former is primarily meant to find tracks that start close to the interaction point (IP) whereas the latter searches for connected hits and is also efficient for displaced vertices. The results from both of these algorithms can be merged to obtain the reconstructed tracks. These tracks are additionally improved by a combinatorial Kalman filter which also takes the SVD data into account. If the CDC does not produce enough hits, a standalone SVD algorithm is used. The found tracks are finally fitted using a deterministic annealing filter of the GENFIT2 package [5] [10].
The global Legendre algorithm is the primary and currently only track finding algorithm used in Belle II. The cellular automaton algorithm exhibits issues with the track finding quality and is therefore switched off. As the Legendre algorithm is used to compare the performance of the CAT Finder and CAT LINK it is discussed in more detail in the following section.

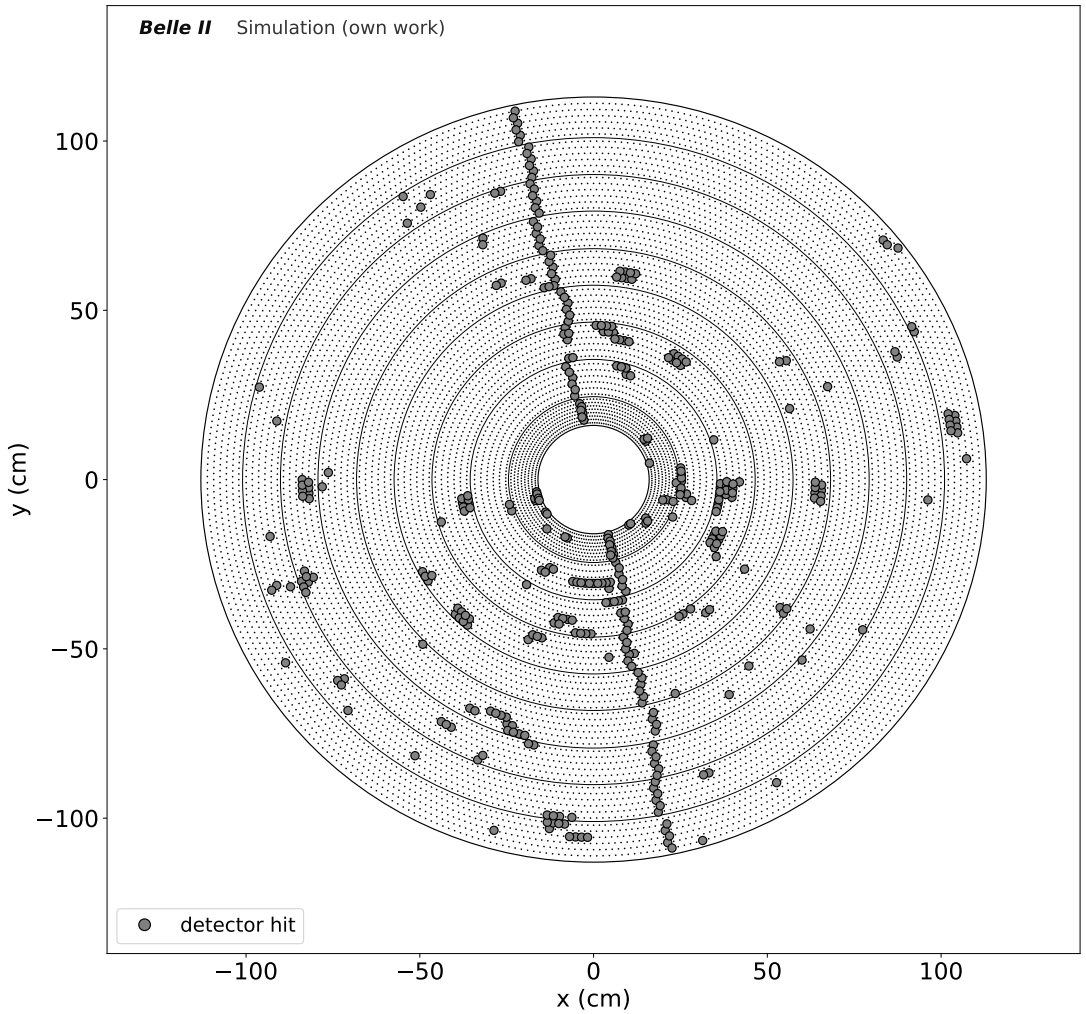


Figure 1.3: Simulation of a CDC measurement for an $e^+e^- \to \mu^+\mu^-$ event with low beam-background. The plot shows the CDC cross-section with the wire hits highlighted. The two tracks from the μ^+ and μ^- are visible along with some hits produced by beam-background. Image provided by Lea Reuter.

Legendre track finding

This global track finding algorithm is designed to find tracks with origins close to the IP. During data taking, a time to digital converter (TDC) records the time between the event trigger signal and the signal of each sense wire. This time is then used to calculate drift circles which give all positions at which the particle could have crossed the sense wire. These drift circles are now used as an input to the Legendre algorithm. As the name of the algorithm implies it applies a Legendre transformation from the r-φ-plane to a conformal space which represents the curved tracks as straight lines while the drift circles remain circles. In the beginning, this is only done for the axial layer hits. The problem now is to find tangents to the drift circles that intersect the origin. The tangents can be expressed by

$$\rho = x_0 \cos\theta + y_0 \sin\theta \pm R_{\mathrm{dc}} \tag{1.1}$$

with the Legendre parameters ρ and θ, x- and y-coordinate x_0 and y_0 and radius R_{dc} of the drift circle. Viewing this in the ρ-θ-space produces a sinusoid for each drift circle.

Finding the most populated areas in this space yields the Legendre parameters for the most optimal tangent. Applying an inverse Legendre transformation to this tangent then finally returns the track. To do this, the ρ-θ-space is split into four equally sized bins. The most populated bin is selected and the process is repeated until the bins are smaller than a ρ-dependent resolution parameter. Figure 1.4 shows this technique.
For multiple tracks in the CDC, this track finding process is repeated while different track candidates are investigated in each iteration. As the next steps, a slightly modified process is employed which allows for tracks with origins offset from the CDC center to be found. Additionally, the stereo layer hits are used to gain z-axis information [5].

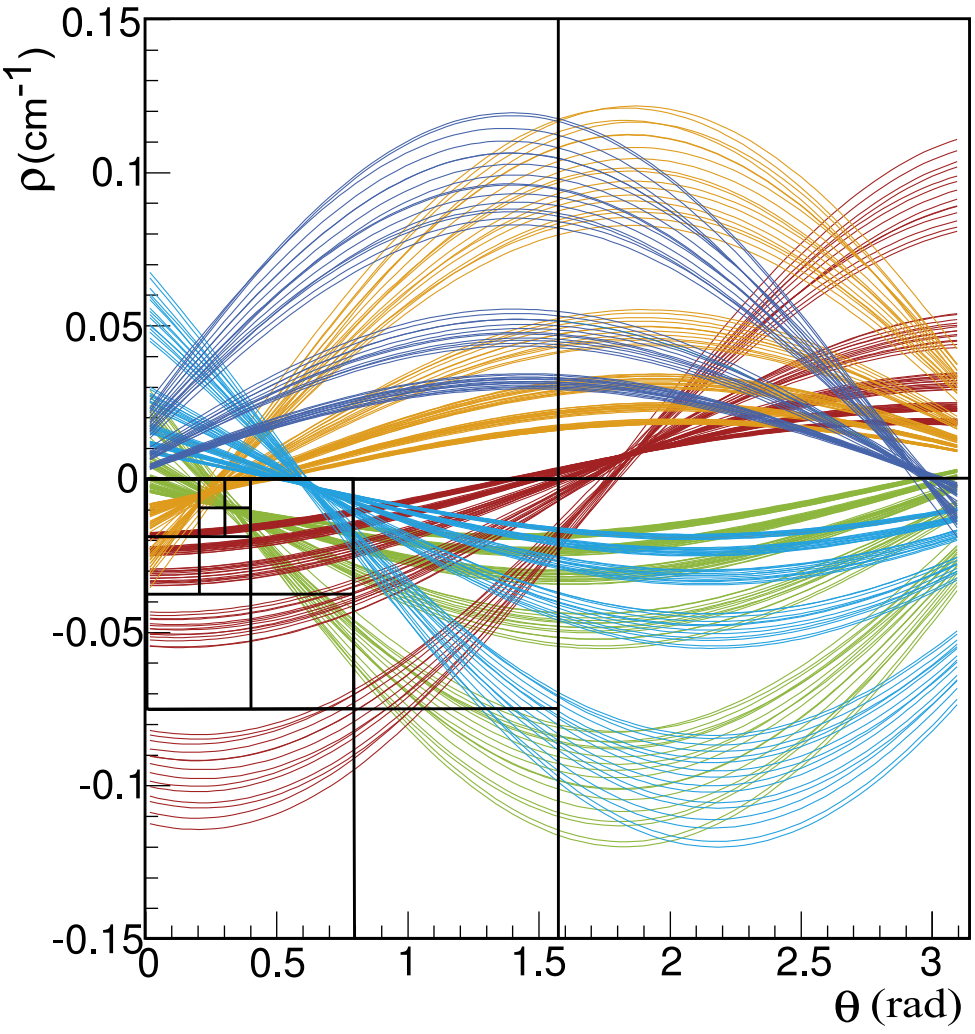


Figure 1.4: Bin subdivision in Legendre tracking. The bins are quartered and the most populated bin is selected. This process is repeated until a given resolution parameter is reached. Image taken from [5].

CAT Finder

As the Legendre track finding algorithm is currently the only track finding algorithm used, the track finding efficiency for tracks that originate further from the IP is accordingly poor. This is problematic since these displaced vertices are an important signature in dark sector searches like, for example, the production of dark Higgs [11]. A possible solution to this is the development of a new track finding algorithm that also efficiently detects displaced tracks. CAT Finder, which is currently under development, aims to achieve this goal by using graph neural networks (GNN) [12] and object condensation [13]. This technique transforms the hits of the CDC from the real space to a latent space. The CDC hits that correspond to a track condense into a cluster while background hits tend to isolate themselves. It is thought to work well with an unknown number of tracks while also adhering to computing resource constraints [2].

Figure 1.5 shows the architecture of the GNN used for the CAT Finder. An input matrix (green) represents the CDC hit data and is fed to the GNN (blue). The CDC hit data consists of the coordinates of the hit wire and the signal height (ADC) and time (TDC) for each hit. The GNN itself consists of multiple layers, each including a GravNet layer. The GNN ultimately outputs (orange) the cluster coordinates of the object condensation and a beta value, which is a weight between 0 and 1 assigned for every hit. The GNN also outputs the parameters of the tracks that were predicted from the CDC hits. These parameters are the starting position and momentum of the track. By applying a selection criterion on the beta value, the condensation points are found and the hits within a certain range in the latent space of the GNN are assigned to predicted tracks [2] [14]. The predicted tracks are then stored by the Belle II software and passed to following reconstruction steps (e.g. track fitting).

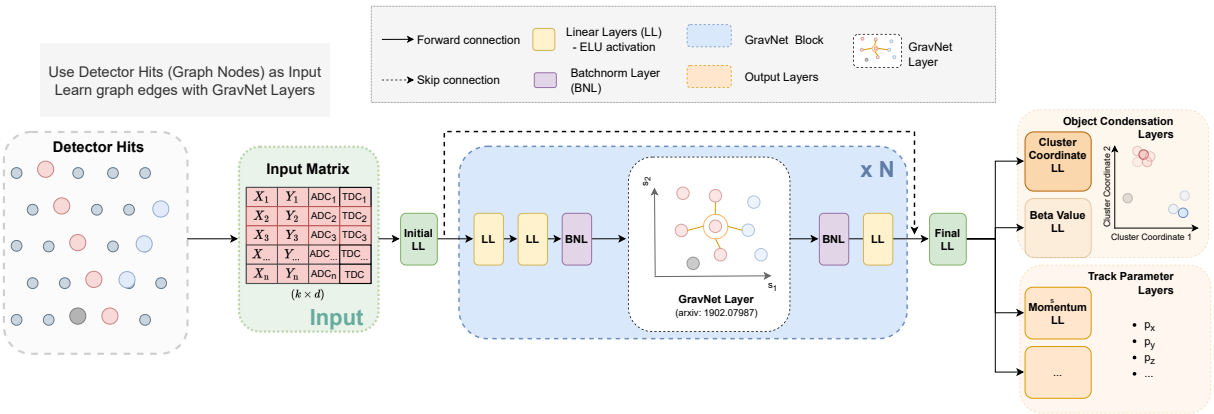


Figure 1.5: Architecture of the CAT Finder GNN. From left to right, the CDC hits are stored in an input matrix which is passed to the GNN. The GNN outputs the object condensation and track parameters. Image taken from [14].

1.3 Motivation for the C++/Python interface

The main problem that arises is that, unlike the established tracking algorithms, the CAT Finder is purely written in Python. While this brings the advantage of faster prototyping and better readability, lower performance in runtime and memory consumption are a drawback. While it may seem simple to translate a stable version of the Python code to C++, this is not easily possible. The basf2 software currently does not support LibTorch, the C++ equivalent of PyTorch [15] which is used the by CAT Finders GNN, as an external library. The solution is to do as many of the calculations as possible in C++ and interface the data to and from the Python GNN. The software that performs these calculations and provides the interface will be referred to as the **CAT L**anguage **I**nterface for **N**eural Networ**k** (CAT LINK).
The main goals of CAT LINK are faster runtime and lower memory consumption compared to the standalone CAT Finder while producing the same results.

Chapter 2

Implementation of the C++/Python interface

2.1 The basf2 environment

The simulation and analysis of particle collision events is done using the Belle II Analysis Software Framework (basf2), an analysis framework developed by the Belle II experiment [1] [16].
The framework contains several modules which can be loaded dynamically by the software. The functionality of these modules can be divided into core modules and analysis modules. While the latter perform the actual analysis of the events, core modules can be used to steer and monitor the basf2 framework itself. Examples for basf2 modules are given in Table 2.1. All modules inherit from the `Module` base class, which offers interface methods that are called during initialization or termination of the analysis, before or after a run is processed, or during each processed event. The modules in which performance is crucial are written in C++ although Python modules are also used [1].

Table 2.1: Example modules of basf2 [17].

Module	Description
`EventInfoSetter`	Mandatory module to set metadata for the current event.
`Progress`	Prints the number of processed events to the standard output.
`RootOutput`	Saves objects from the global data store to a `.root` file. The data can be loaded using the `RootInput` module.
`StatisticsSummary`	Provides a summary of the statistics of all modules that are called between two `StatisticsSummary` modules.
`CATFinder`	Module containing the novel track finding algorithm studied in this thesis.

Another important component of a basf2 analysis is the `Path` object. Modules can be added to it and are then executed when calling the `process()` method. Each event undergoes the whole sequence of modules in the order that they are added to the `Path` instance.
A Python interface is provided by basf2 that allows the path and its modules to be configured in Python files called "steering files". Custom modules can also be defined here. Steering files may be executed directly by a Python interpreter or by using the basf2 executable. Using the executable allows for framework-specific arguments, e.g. the number of events that should be simulated, to be passed [1].
The framework also contains a globally accessible data store. The data store provides access to objects and arrays of objects that are saved in `StoreObjPtr` or `StoreArray` objects respectively. The data store can also store many-to-many relations between its objects. This can for example be used to relate an object to other objects that were used to create it.

In addition to the basf2 framework, the Belle II software also relies on the use of "externals" which are third-party applications not specific to Belle II. These externals are distributed with basf2 and, among others, include a C++ compiler, Python interpreter, and high-energy physics related software such as Geant4 [18] and EvtGen [19]. In total, there are about 60 external packages as well as 90 Python packages [1] [20]. Externals bring extremely powerful tools for analysis in Belle II and the lack of appropriate externals can hinder the development of new modules as discussed in Section 1.3.

2.2 Methods of code optimization

Optimizing code strongly depends on the application the code is supposed to serve. There are however several practices that can be applied to any project, and often across programming languages, to ensure better code performance. Some practices that are relevant for CAT LINK are:

- Avoiding local variables.
  Removing unnecessary variable definitions not only clears memory but can also save the time that would be needed to access this memory space.
- Avoiding casting.
  Casting often includes copying data in the background and performing conversions that could be avoided by using an appropriate datatype to begin with.
- Avoiding dynamic memory allocation.
  While this feature is very useful to have, it requires the system to find a suitable spot in the memory to save object data every time the object grows.
- Optimizing loops.
  It might be possible to terminate some loops as soon as a specific outcome is reached. Loops could also be replaceable by optimized functions from third-party packages.

Additionally, some code optimization can also be done automatically by the C++ compiler.
A problem specific to CAT Finder and CAT LINK is the PyROOT C++/Python interface provided by the ROOT data analysis framework [21]. This interface is built robust but is not optimized for performance. As the data is handled by C++, accessing the data via Python might lead to unknown data conversion operations behind the scenes. Implementing specialized methods that handle these operations allows for a better understanding of the internal processes of the interface and optimize it to suit the performance requirements of the CAT Finder.

2.3 Implementing the C++/Python interface

The CAT Finder basf2 module can conceptually be split into three parts: preprocessing, running the GNN, and postprocessing. The preprocessing stage accesses the CDC hits provided by the basf2 software and extracts relevant information. This is the x- and y-coordinates of the center of the wire hit, TDC and ADC counts as well as the CDC layer and superlayer. The preprocessing stage then converts this information into pure Python objects (e.g. lists) that are accepted as input by the GNN. After running the GNN, the postprocessing stage accepts its output and reconstructs the track from it.

The concepts of code performance improvements discussed in Section 2.2 are now applied to the pre- and postprocessing stages. The CAT Finder module will remain in Python with calls to the C++ code at the appropriate spots.

2.3.1 Preprocessing

As a first naive implementation, the Python code is replicated in C++ with no regard for performance.
The preprocessing starts by retrieving all of the CDC hits and preparing Python lists with a size that equals the number of CDC hits. Looping over all CDC hits, the information about the current hit is saved, which includes the CDC global layer number, wire, wire position, TDC, and ADC counts. The TDC and ADC information then has to be scaled to a range of -1 to 1 and 0 to 1 respectively. They are then saved to the Python lists using `PyList_SetItem()`[22]. The layer and wire information is used to determine the x- and y-position of the hit which are also saved to Python lists. The two final Python lists are filled with the CDC layer and superlayer.

With a functional prototype in place, improving the performance is now focused. Firstly, the GNN is still written in Python and as such expects Python objects to be passed to it. None of the arrays are needed as C++ vectors. It is therefore better to save the CDC hit data directly into Python lists instead of creating a C++ vector and converting it to a Python list. Since the preprocessing stage only retrieves data from the basf2 data store and performs some basic calculations on it, this is already the highest impact the optimization efforts have. Nonetheless, some minor improvements are added in a second iteration. These are the removal of unnecessary variable definitions and the strategy for handling the Python lists is revised.

As the length of each list is equivalent to the amount of CDC hits, lists of a fixed size are prepared beforehand to avoid dynamic memory allocation. Additionally, instead of creating new list objects for each event, the existing lists are emptied and reused.

The effects on the average per-event runtime of the interface are shown in Figure 2.1. Optimization step 0 refers to the first prototype, step 1 is after storing the data in Python objects and step 2 includes the minor improvements as mentioned earlier.

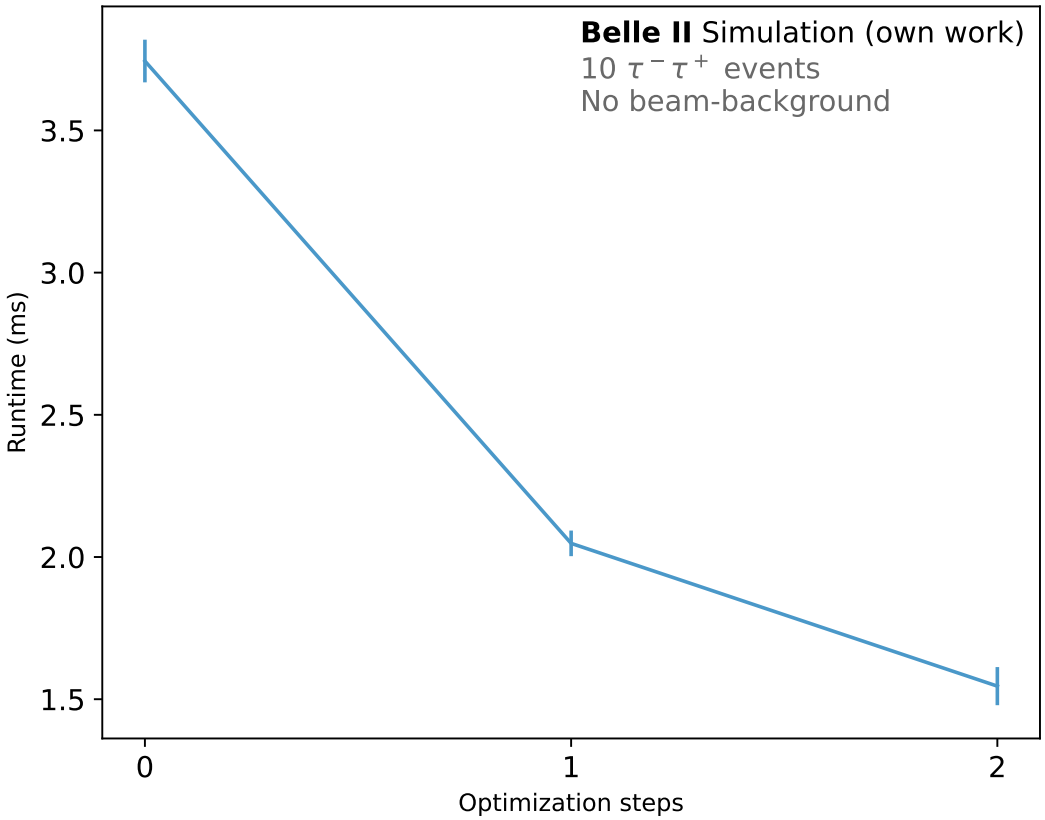


Figure 2.1: Runtime performance of the preprocessing stage after optimization. Step 1 refers to the implementation of Python lists and step 2 is other minor improvements as discussed above.

2.3.2 Postprocessing

For the postprocessing stage, the same basic approach of converting the Python code into C++ code and then iteratively optimizing it is used. As the GNN returns Python objects, they need to be converted to C++ objects first. For primitive datatypes, the corresponding functions provided by the Python/C API [22] are used, such as `PyFloat_AsDouble()`. For converting the Python lists, a new function is defined as follows:

First a `std::vector<double>` object is created that will store the Python list. After using `PyList_check()` to make sure that the GNN output is a list, memory is reserved as needed according to the length of the list. In a loop, every entry of the Python list is retrieved using `PyList_GetItem()`, checked that it is a float with `PyFloat_Check()` and, if so, appended to the C++ vector.
A second function that converts a Python list to a `std::vector<long>` C++ object is defined in the same fashion.

Now the GNN output can be accessed as C++ objects and the reconstruction is done by looping over all of the condensation points. Within this loop, the distance of each hit to the current condensation point is calculated in the latent space. If this distance is smaller than a previously defined threshold d_{hit}, the hit index is saved to a list. For all of the simulations discussed later, the threshold is set to $d_{\text{hit}} = 0.5$.
The momenta and positions of the track can be retrieved from the `vec_con_point_px`, `vec_con_point_vx`, and respective y- and z- lists. This information is now added to a newly created `RecoTrack` object, which is in turn added to the data store. The `RecoTrack` object is also equipped with a covariance matrix that describes the uncertainties of the predicted track using default values. Finally all of the previously found CDC hits that are close enough to the condensation point are linked to the reconstructed track and the loop continues with the next condensation point.

Chapter 3

Performance analysis

3.1 Methodology

To verify that the new basf2 module satisfies the expectations mentioned in Section 1.3, the CAT LINK C++/Python interface is extensively tested. This means comparing the CAT LINK results in several benchmarks to the results of the pure Python CAT Finder subjected to the same benchmark conditions. Additionally, the tests are also performed with basf2's Legendre algorithm to compare the novel algorithms with an established one.

As the main goal of CAT LINK is to speed up track reconstruction in the CAT Finder, the runtime performance is the most critical metric to discuss for the fully functional interface. Here the runtime performance refers to the average time needed for the algorithm to reconstruct simulated Monte Carlo particles (MC particles) from hits in the CDC of a single event. This event runtime is further broken down into the runtimes of the preprocessing, GNN, and postprocessing stages. Since the Legendre algorithm is fully implemented in C++, the runtime is not broken down.

Another important metric to consider is the memory consumption. A detailed memory profiling proves to be more difficult than the analysis of the runtime performance.

To verify that the reconstructed tracks delivered by CAT LINK match those found by the CAT Finder, the track finding efficiency, fake rate, and relative transverse momentum resolution (p_t resolution) are measured according to the truth transverse momentum.
For the track finding efficiency and fake rate, the particles that were reconstructed using the tracking algorithm and the particles that were actually simulated need to be saved. For the p_t resolution, the predicted and simulated momenta of each particle are also looked upon. To understand how CAT LINK behaves with different event conditions, all benchmarks are performed for four different event types:

- $e^+e^- \to \mu^-\mu^+$ (generated using KKMC [23])
- $e^+e^- \to \tau^-\tau^+$ (generated using KKMC and Tauola [24])
- $e^+e^- \to B^0\overline{B^0}$ (generated using EvtGen [19])
- $e^+e^- \to B^+B^-$ (generated using EvtGen)

and three different beam-background scenarios [3]:

- No beam-background
- Low beam-background corresponding to the average SuperKEKB conditions in Run 1
- High beam-background corresponding to the average SuperKEKB conditions at the peak design luminosity

30 000 events are simulated for each combination of tracking algorithm, event type, and beam-background scenario. The same random seed was used for all simulations. Some of the data was lost due to an unknown bug in the simulation workflow. One possible explanation is the occurrence of race conditions as the events were simulated using multiprocessing. The missing data is filled with data from simulations of 3 000 events. Any data that is based on this dataset with lower statistics is marked with a dagger $(\dagger)$ in the plots. The memory performance data is fully based on datasets of 1 000 events. For the simulation, a dedicated basf2 branch detached from commit `cd980c80` is used. The beam-background files are centrally produced by the collaboration using release-06-00-08. As explained in Section 1.2.1, different beam-background conditions and different event types lead to a different number of CDC hits and the track finding algorithm performance varies with the number of CDC hits.

3.1.1 Steering file

The steering file used in all of the benchmarks executes the `CATFinder` module containing the GNN based tracking algorithm, the `RecoTrackInfo` module as well as the standard basf2 simulation path. the `RecoTrackInfo` module is used to determine the tracking efficiency, fake rate, and p_t resolution of the tracking algorithm used in the path. How each of those metrics is calculated within the module is explained in Section 3.2 to Section 3.6.

The steering file can be launched with basf2 and requires the arguments defined in Table 3.1 using the `argparse` Python package [25]. The `--event-type` flag adds the KKMC generator for $e^+e^- \to \mu^-\mu^+$ and $e^+e^- \to \tau^-\tau^+$ events or the EvtGen generator for $e^+e^- \to B^0\overline{B^0}$ and $e^+e^- \to B^+B^-$ events. The beam-background is chosen with the `--background` argument. For high beam-background, the `expList` argument has to be set to 0 when adding the `EventInfoSetter` module to the path. For other beam background scenarios, `expList` is set to 1 003. Lastly, the tracking algorithm that is chosen with the argument `--tracking-algorithm` is added to the path. For the Legendre tracking, the `StatisticsSummary` module has to be added directly before and after adding the Legendre tracking itself. This allows for the memory consumption and per-event runtime to be measured. The steering file saves all of the necessary data to the file path specified with the `--output-file` argument.

Table 3.1: Supported command line arguments for the steering file.

Argument	Accepted values	Description
`--event-type`, `-e`	`mumu`, `tautau`, `BB`, `BBbar`	Defines which event type should be simulated. Currently supported are $e^+e^- \to \mu^-\mu^+$, $e^+e^- \to \tau^-\tau^+$, $e^+e^- \to B^+B^-$ and $e^+e^- \to B^0\overline{B^0}$ events.
`--tracking-algorithm`, `-a`	`leg`, `cat`, `int`	Sets the tracking algorithm. Currently supported are Legendre tracking, CAT Finder and CAT LINK respectively.
`--background`, `-b`	`none`, `low`, `high`	Sets the beam-background scenario.
`--output-file`, `-o`	any filepath	Defines a file to which the data is written. An existing file will be overwritten.

The process of collecting important data is automated for every benchmarked scenario using an appropriate shell script. To go easy on computing resources, the steering files are executed sequentially and not in parallel. At this point, it will also be beneficial to choose file names for the output files that can easily be processed later on. An example command for running the simulation is given here:

```
basf2 -n 30000 -p 50 steering_file.py -- -a int -e tautau -b high
-o int_tautau_high_30000.txt
```

The basf2 argument `-n` sets the number of events that should be simulated in a single run and `-p` sets the number of CPU cores that should be used in parallel. This command is run 36 times with the arguments adjusted accordingly to simulate every possible combination.

3.2 Runtime performance

To measure the runtime performance of the CAT Finder and CAT LINK, the `perf_counter_ns` class from Pythons `time` package [26] is used. Once called, this class returns the value of a performance counter with nanosecond precision. Calling the class again later and subtracting the performance counter value from the first call then gives the elapsed time between both calls in nanoseconds. To be consistent with the basf2 statistics, this runtime is converted to milliseconds. Now the runtime of all of the stages in the CAT Finder event method can be determined by simply keeping track of the performance counter value at the start and end of each stage in the code.

As mentioned before, the runtime of the Legendre algorithm can not be easily retrieved this way. Instead, the basf2 statistics function has to be used. Another challenge is that the Legendre algorithm consists of multiple basf2 modules and as such there is not a single value that corresponds to the Legendre runtime in the basf2 statistics. To solve this, another

module is introduced to the path: the `StatisticsSummary` module. This module will add another entry to the basf2 statistics which contains the sum of all of the modules that have been called between the first and second call of the `StatisticsSummary` module. Adding this module to the path directly before and after adding the Legendre algorithm then yields the per-event runtime of the Legendre algorithm as the output of the `StatisticsSummary` module.

Figure 3.1 shows the per-event runtimes of each tracking algorithm measured using the described methods. It is worth noting that the results of the first event of each simulation are discarded. The initialization of the simulation framework causes this event to have a significantly higher runtime compared to the following events, distorting the average runtime. It is assumed that, in a realistic scenario, the initialization is done before the first event so this performance drop is not expected.
Comparing the CAT Finder and CAT LINK in an example scenario it is evident that the performance of the preprocessing stage increased drastically (4,7 times). For the postprocessing stage, a small performance drop (1.7 times) is observed in this specific scenario. Overall, in both cases, the average event runtime is dominated by the GNN and about 1,7 times larger than for the Legendre track finding algorithm.

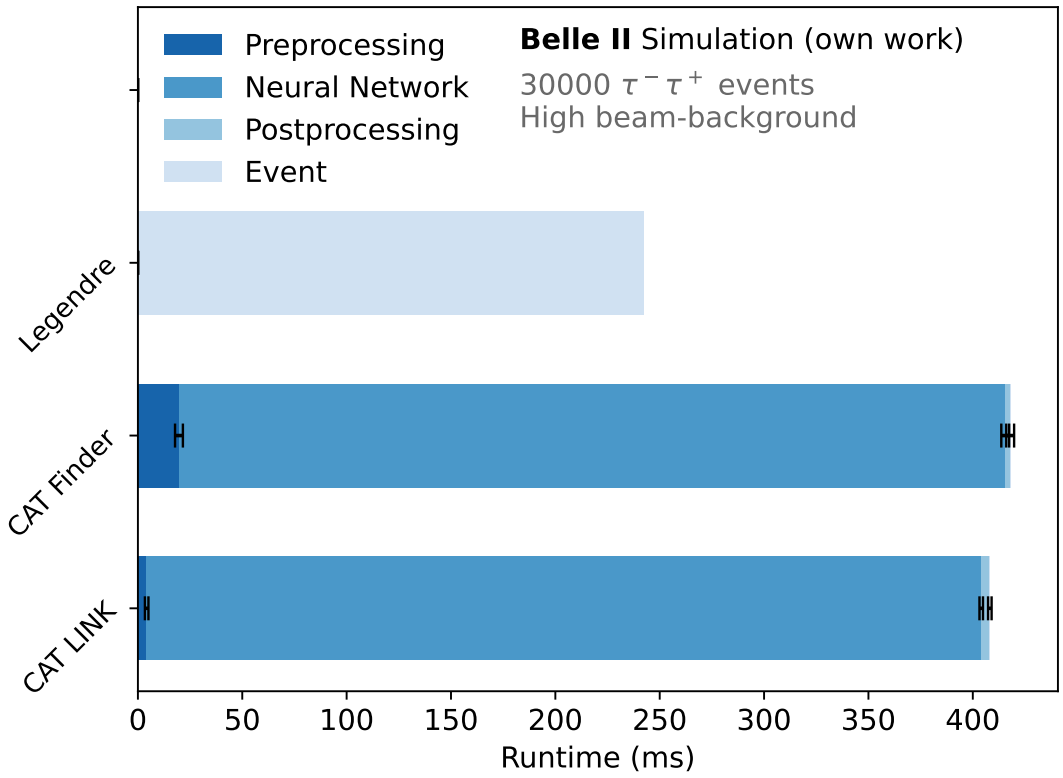


Figure 3.1: Comparison of the per-event runtimes of the different track finding algorithms for 30 000 $e^+e^- \to \tau^-\tau^+$ events with high beam-background. The Legendre algorithm shows the fastest runtime, CAT Finder and CAT LINK being roughly 1,7 times slower. The preprocessing runtime is significantly reduced while the postprocessing runtime slightly increased in this specific scenario. Overall, CAT LINK performs slightly better than CAT Finder.

3.3 Memory performance

Measuring the memory consumption of all of the stages separately is not as easy as the runtime. Therefore, the memory profiling provided internally by basf2 has to be relied upon. This comes with the disadvantage of low resolutions of approximately $\mathcal{O}(1\,\mathrm{MB})$. Additionally, the memory consumption as measured by basf2 is easily influenced for example by `print()` commands in the code or concurrent activity on the machine. The memory consumption of a module provided by basf2 is only for the entire runtime, not per event. All in all, it is clear that the absolute memory values are not reliable but are still useful to compare the different algorithms.

Figure 3.2 shows a slight drop in memory consumption when using CAT LINK compared to the CAT Finder (1.2 times) though both algorithms are outperformed by the Legendre tracking. This proves that the internal implementation in C++ provides better memory management for CAT LINK.

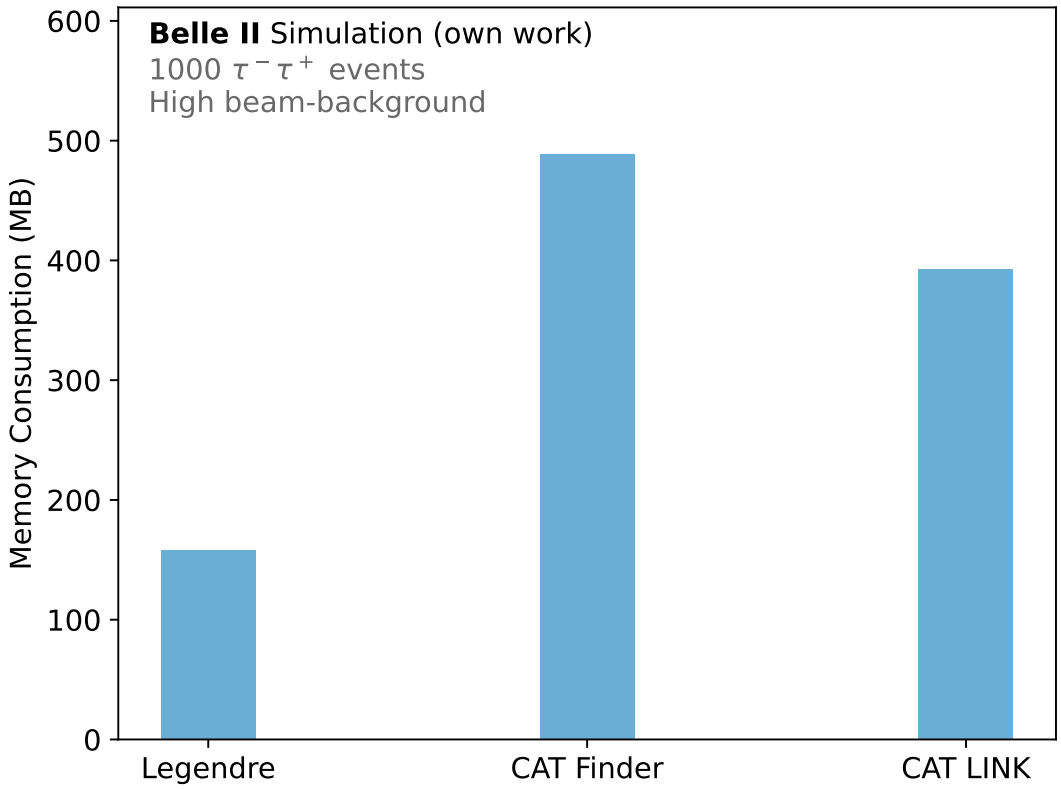


Figure 3.2: Comparison of the memory consumption of the different tracking algorithms based on 1000 $e^+e^- \to \tau^-\tau^+$ events with high beam-background. The Legendre algorithm again exhibits the best performance while CAT LINK performs about 1,2 times better than the CAT Finder.

3.4 Track finding efficiency

As a first test to see if CAT LINK exhibits the same performance as the CAT Finder alone, the track finding efficiency is determined. It is defined as

$$\eta_{\text{track}} = \frac{N_{\text{found}}}{N_{\text{MC}}}, \tag{3.1}$$

where N_{found} is the number of simulated tracks that were found by the tracking algorithm and N_{MC} is the total amount of simulated tracks. Both N_{found} and N_{MC} are implicitly stored as the length of the `MCParticles` and `RecoTracks` store arrays in basf2. These are the names of the containers for the simulated particles and the tracks found by the tracking algorithm respectively. All necessary calculations are defined in a dedicated basf2 module which can then be used for all tracking algorithms.
For each of the found tracks, the predicted momenta p_{x}, p_{y} and p_{z} of the track are determined and used to calculate the transverse momentum as

$$p_{\text{t}} = \sqrt{p_{\text{x}}^2 + p_{\text{y}}^2}. \tag{3.2}$$

This information is used to fill a histogram with 30 bins of 200 MeV width, ranging from 0 GeV to 6 GeV, by increasing a counter for the bin corresponding to the predicted momentum. In the next step, a similar histogram is created by repeating this process for the simulated particles. The track finding efficiency can then be determined in relation to the transverse momentum. Figure 3.3 shows the resulting plot for $e^+e^- \to \tau^-\tau^+$ high beam-background events. The x-axis does not span the full 6 GeV range as no corresponding particles are generated or found above a certain momentum. The plot shows that the track finding efficiencies of the CAT Finder match perfectly with the efficiencies of CAT LINK. It is also clearly visible that the CAT Finder and CAT LINK yield at least the same efficiency as Legendre for $p_{\text{t}}^{\text{MC}} \geq 0.6$ GeV and in most cases even surpass it.

3.5 Fake rate

As the next metric, the fake rate is investigated. The fake rate of a tracking algorithm describes the ratio $\overline{N}_{\text{found}}$ of how many of the predicted tracks do not originate from a simulated particle to the total number of simulated particles. It is therefore defined as

$$\eta_{\text{fake}} = \frac{\overline{N}_{\text{found}}}{N_{\text{MC}}}. \tag{3.3}$$

The module created for the track finding efficiency is expanded by adding another histogram for counting with the same properties as before. When looping over the found tracks it is checked if the track is related to a simulated particle. If not, the counter of the bin corresponding to the predicted p_{t} is increased. Figure 3.4 shows the resulting plot. The fake rates for the CAT Finder and CAT LINK again match perfectly. For $p_{\text{t}}^{\text{MC}} \leq 4.6$ GeV slightly higher fake rates are observed for the GNN-based algorithms with higher deviations for $p_{\text{t}} \leq 2$ GeV. It is worth noting that results in this area might not be as trustworthy due to lower statistics.

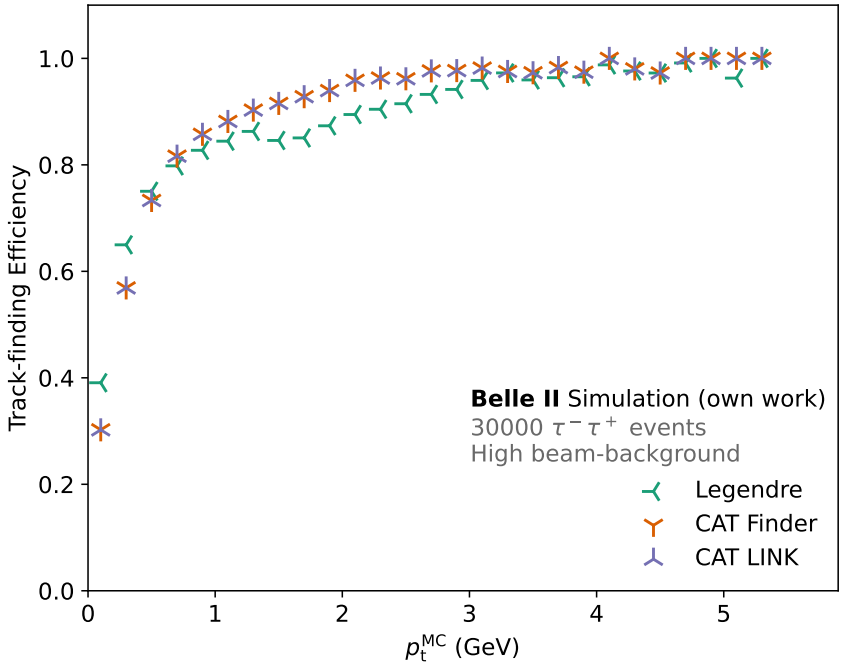


Figure 3.3: Comparison of the track finding efficiency of the different tracking algorithms based on 30 000 $e^+e^- \to \tau^-\tau^+$ events with high beam-background. The results of the CAT Finder (orange) and CAT LINK (purple) match for $p_t^{MC} \geq$ 0.6 GeV and in most cases outperform the Legendre algorithm (green). The step size of p_t^{MC} is set to 0.2 GeV.

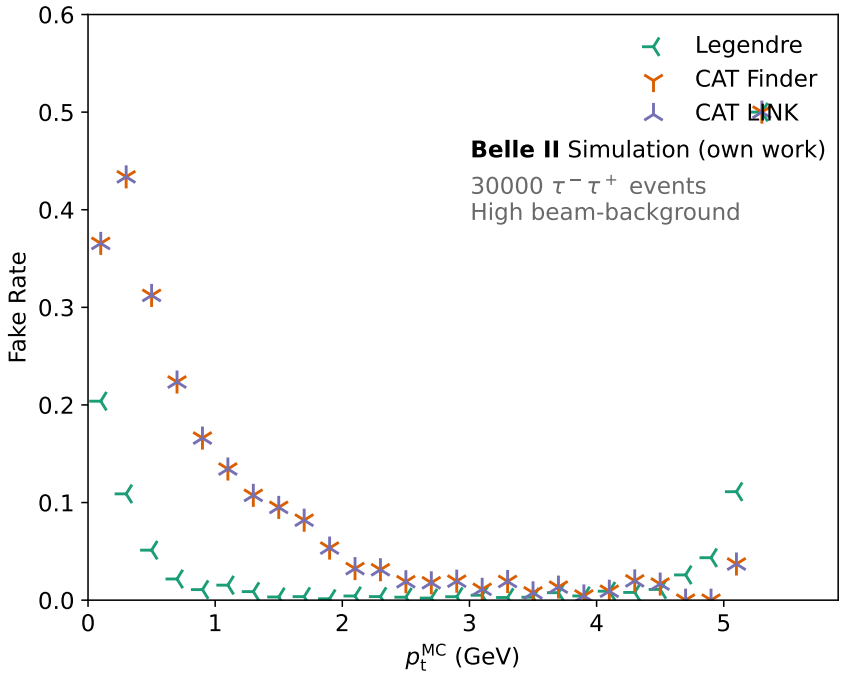


Figure 3.4: Comparison of the fake rate of the different tracking algorithms based on 30 000 $e^+e^- \to \tau^-\tau^+$ events with high beam-background. The CAT Finder (orange) and CAT LINK (purple) results again match over the whole transverse momentum range. The Legendre algorithm (green) outperforms the GNN-based algorithms for $p_t^{MC} \leq$ 4.6 GeV. The step size of p_t^{MC} is again set to 0.2 GeV.

3.6 Resolution of transverse momentum

For the p_t resolution, the tracks found by the tracking algorithm that are matched to a simulated particle are studied. The transverse momentum resolution is defined as the full width at half maximum (FWHM) of the distribution given by

$$f\left(p_\mathrm{t}^\mathrm{pred}, p_\mathrm{t}^\mathrm{MC}\right) = \frac{p_\mathrm{t}^\mathrm{pred} - p_\mathrm{t}^\mathrm{MC}}{p_\mathrm{t}^\mathrm{MC}} \tag{3.4}$$

As the resolution should again be in relation to p_t^MC, a histogram of the distribution in Equation (3.4) is created for every p_t^MC bin. These have a range from −0.15 to 0.15 with 120 bins. To populate these resolution histograms, it is first checked which p_t^MC bin they belong to. Afterward, the predicted and simulated momenta are plugged into Equation (3.4) and the counter of the resulting bin is increased. Once this is done for all reconstructed tracks, a fit is performed on each of the histograms using the Crystal Ball probability density function [27] and the FWHM of the fitted function is calculated. The model function is

$$f_\mathrm{m}\left(x, x_0, A, \alpha, n, \overline{x}, \sigma\right) = A \cdot f_\mathrm{cb}\left(x - x_0, \alpha, n, \overline{x}, \sigma\right) \tag{3.5}$$

with the x-offset x_0, amplitude A, and Crystal Ball function f_cb defined as

$$f_\mathrm{cb}\left(x, \alpha, n, \overline{x}, \sigma\right) = N \cdot \begin{cases} \exp -\frac{x^2}{2} & \text{if } x > -\alpha \\ \left(\frac{n}{|\alpha|}\right)^2 \exp\left(-\frac{\alpha^2}{2}\right)\left(\frac{n}{|\alpha|} - |\alpha| - x\right)^{-n} & \text{if } x \leq -\alpha \end{cases}. \tag{3.6}$$

A Crystal Ball function describes a distribution with a Gaussian core and power-law tails. β defines the point at which the distribution changes from a power-law to a Gaussian distribution with mean $\overline{x}$ and deviation σ. n is the free parameter of the power-law term and N is a normalization constant. The probability density function of the Crystal Ball function is used from `scipy.stats.crystalball` [28].

An example of this for $p_\mathrm{t}^\mathrm{MC} = 0.6$ GeV is shown in Figure 3.5. The full p_t resolution for all p_t^MC bins is shown in Figure 3.6. Any gaps in the data can be explained by a failed Crystal Ball fit which occurs more frequently for the Legendre track finding algorithm. The reason for this is to be investigated. The plot again shows that the CAT Finder and CAT LINK results match, albeit with worse results than for the Legendre algorithm.
Combined with the results from Section 3.2 and Section 3.3 this concludes that CAT LINK exhibits the same performance as CAT Finder while reducing both runtime and memory consumption, reaching the targets set in Section 1.3.

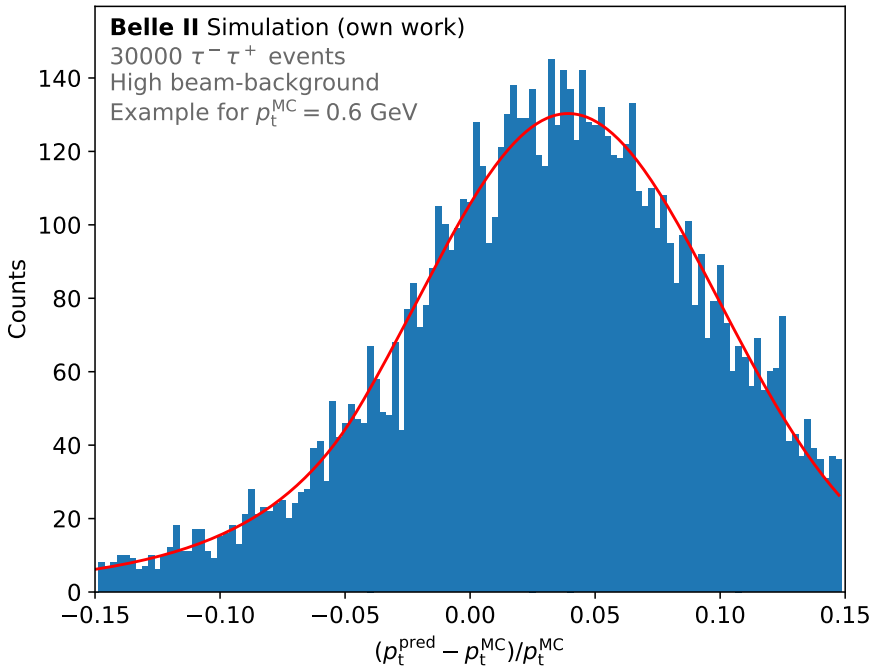


Figure 3.5: Example of a Crystal Ball fit (red) to determine the p_t resolution. The data (blue) relies on 30 000 $e^+e^- \to \tau^-\tau^+$ events with high beam-background. The bin size is set to $2{,}5 \cdot 10^{-3}$.

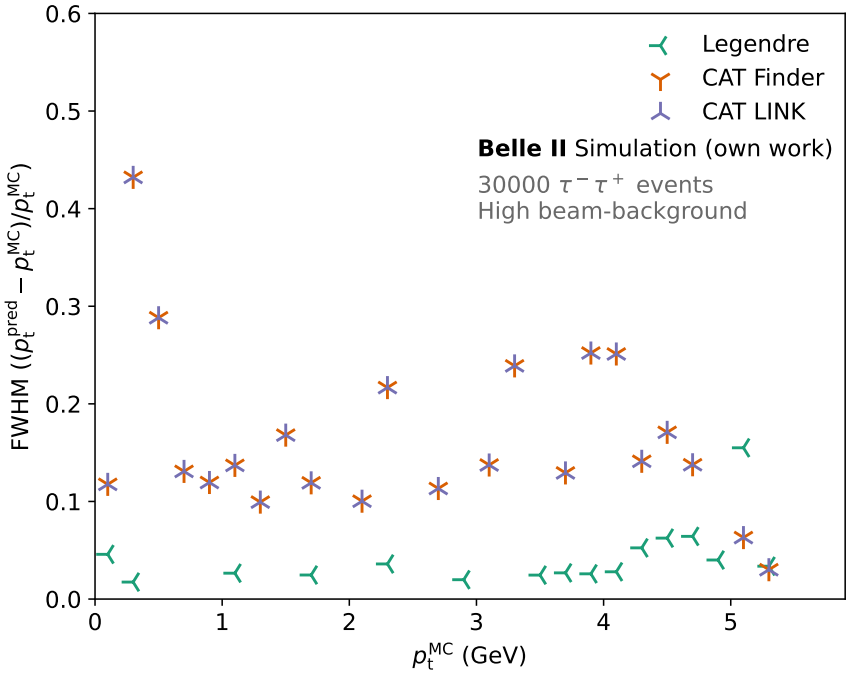


Figure 3.6: Comparison of the p_t resolution of the different tracking algorithms based on 30 000 $e^+e^- \to \tau^-\tau^+$ events with high beam-background. The CAT Finder (orange) and CAT LINK (purple) results again match at every point. For $p_t \leq 5$ GeV the Legendre (green) algorithm performs better than the GNN-based algorithms.

Chapter 4

Conclusion

4.1 Results

All in all, CAT LINK reaches its goals and successfully reduces the per-event runtime and memory consumption of the CAT Finder tracking algorithm. The results of all of the benchmarks can be found in Appendix A.2.

Comparing the per-event runtimes, CAT LINK shows an improvement to CAT Finder especially for events with a low number of CDC hits. For a larger amount of hits, the Legendre algorithm performs worse and its per-event runtime approaches the per-event runtimes of the GNN-based algorithms. For some of the simulations, CAT LINK performs worse than CAT Finder. These performance losses mainly stem from the execution of the GNN which is not altered for CAT LINK. A possible explanation would be that the runtime is influenced by concurrent processes running on the machine.
For memory consumption, CAT LINK surpasses the CAT Finder in every scenario and in some cases also the Legendre algorithm. CAT LINK therefore meets its goal of reducing memory consumption. As described in Section 3.3, external factors easily influence memory consumption and can cause severe fluctuations. A possible solution to mitigate this effect would be to take the average of many benchmarks which would be very time-consuming and was therefore not done here. Another solution would be to run the simulation in a more isolated environment, like a dedicated machine.
The track finding efficiency, fake rate and p_t resolution of the CAT Finder and CAT LINK mostly match except for events without beam-background and $e^+e^- \to B^+B^-$ events with low beam-background. The reason for this discrepancy is to be studied in the future. Nonetheless, the results CAT LINK delivers are close to the CAT Finder results and still within an acceptable range.

4.2 Outlook

4.2.1 Transfer to other parts of the detector

The use of GNNs for the reconstruction of events is also studied for other detector elements in Belle II, such as the electromagnetic calorimeter. It consists of 8 736 thallium-doped CsI crystals that are arranged around the beamline on a cylinder barrel and in forward- and backward-facing endcaps. Photodiodes attached to the back of these crystals measure the energy deposited in the crystals by energetic photons. A photon typically spreads its energy over a volume of up to 5×5 crystals. The reconstruction algorithm now has to find clusters of crystals that contain the energy of a single photon but not from other particles or beam-background [29].
As the GNN developed for this application is based on Pytorch Geometric [30], the same problem as for the CAT Finder arises when implementing the reconstruction algorithm in basf2. As the C++/Python interface developed in this thesis has been proven to benefit the performance of the tracking-algorithm, the same concept could be applied to the reconstruction of photons in the ECL.

4.2.2 Neural network improvements

As shown in Section 3.2, the runtime of the CAT Finder and CAT LINK is still highly dominated by the runtime of the GNN. It seems natural that performance upgrades in this sector would overall benefit the tracking algorithm the most. One possible approach would be to optimize the remaining Python code of the CAT Finder. This would mainly apply to the part of the `event()` method in which the GNN is executed. As most of this is already outsourced to C++, this approach might be challenging and not yield notable results.
Another method would be to revise the GNN itself. Specifically the graph construction using nearest neighbors, which is currently based on [12], is investigated for optimizations.

4.2.3 CAT Finder as C++ module

Since the GNN can not be implemented in C++ at this point (see Section 1.3), another solution would be to implement the CAT Finder module in C++ and interface to the Python GNN. One benefit of this would be that more calculations can be done in C++, potentially reducing the runtime. On top of that this implementation would only require to access the interface once instead of twice which could also benefit the runtime performance. The C++ module would also be relatively easy to implement as a large part of the code developed as part of this thesis can be reused. Additionally, the C++/Python interface has been proven to satisfy the functionality and performance requirements needed for track finding in Belle II using CAT Finder.

Appendix A

Appendix

A.1 Source code

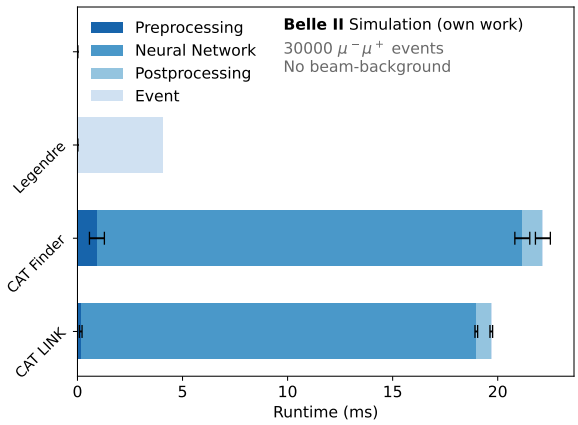
The source code of the CAT LINK interface and the steering file described in Section 3.1.1 are available at `https://zenodo.org/records/10895311` [31].

The C++ code for the pre- and postprocessing is based on the original CAT Finder Python code by Lea Reuter, who also provided the extraction of the momenta of the reconstructed tracks and the simulated MC particles in the `RecoTrackInfo` module.

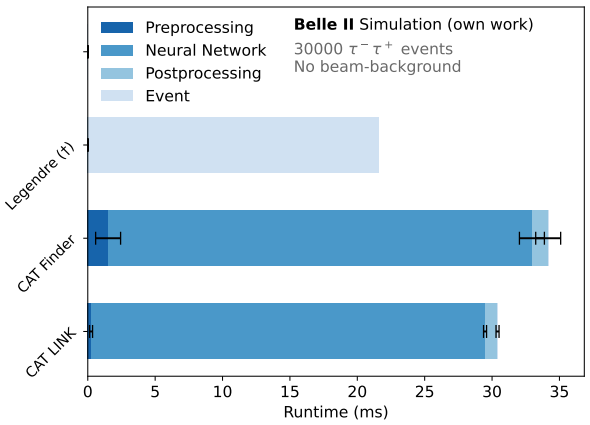
A.2 Plots

The following pages contain the data for all of the benchmarked scenarios as discussed in Section 3.1. The plots are sorted with increasing beam-background downwards and different event types for each column in the order $e^+e^- \to \mu^-\mu^+$, $e^+e^- \to \tau^-\tau^+$, $e^+e^- \to B^+B^-$, $e^+e^- \to B^0\overline{B^0}$.

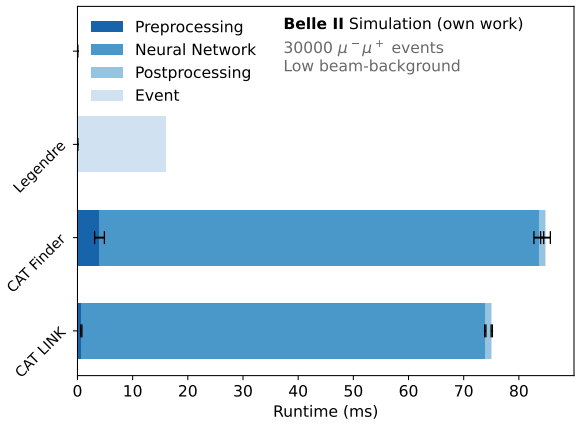
A.2.1 Runtime performance



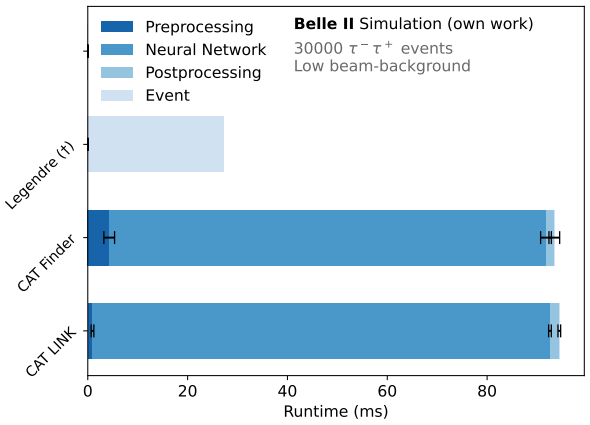
$e^+e^- \to \mu^-\mu^+$ events, no background.



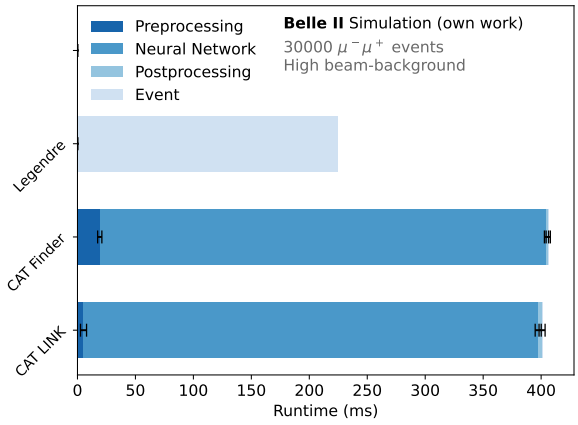
$e^+e^- \to \tau^-\tau^+$ events, no background.



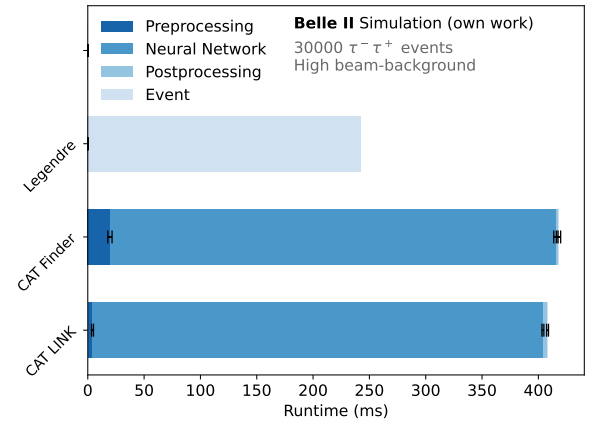
$e^+e^- \to \mu^-\mu^+$ events, low background.



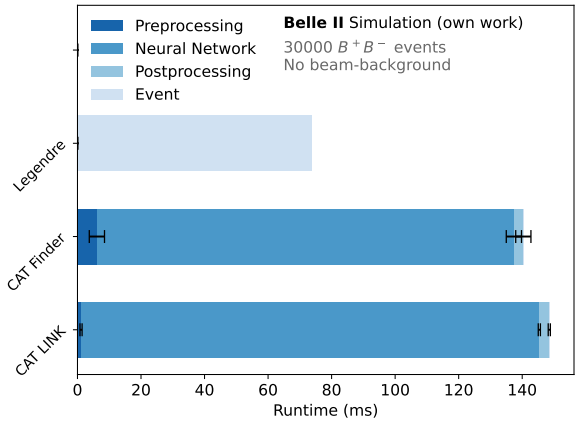
$e^+e^- \to \tau^-\tau^+$ events, low background.



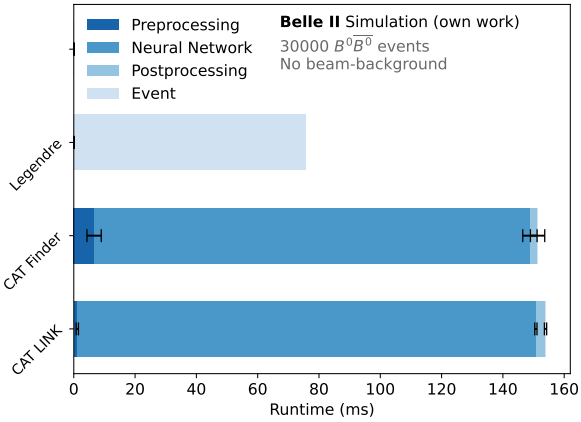
$e^+e^- \to \mu^-\mu^+$ events, high background.



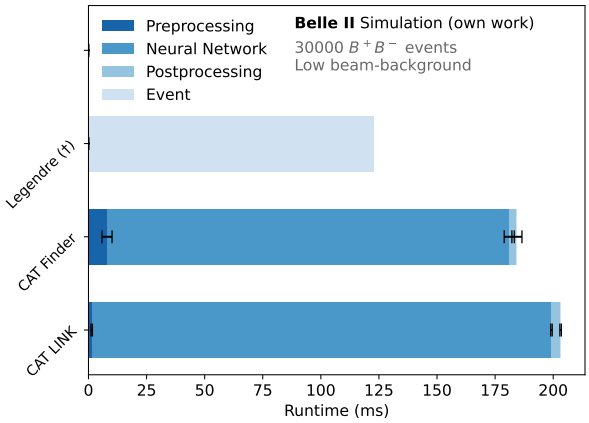
$e^+e^- \to \tau^-\tau^+$ events, high background.



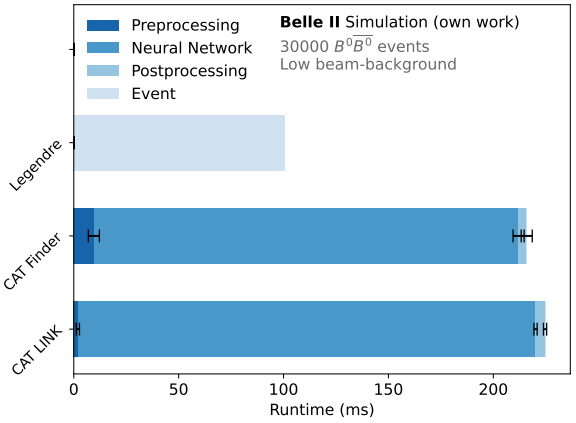
$e^+e^- \to B^+B^-$ events, no background.



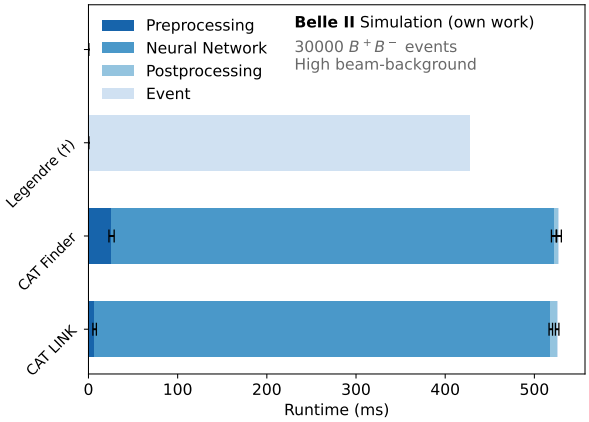
$e^+e^- \to B^0\overline{B^0}$ events, no background.



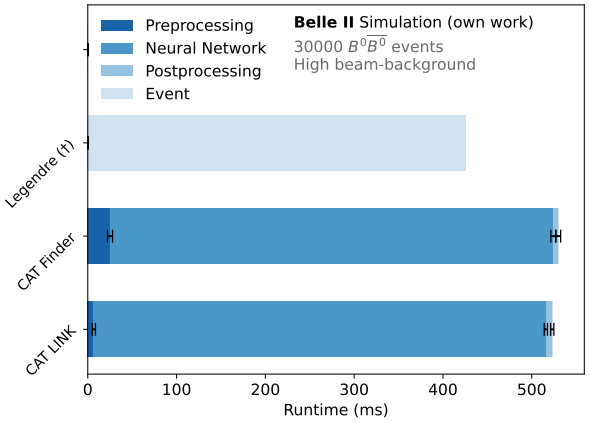
$e^+e^- \to B^+B^-$ events, low background.



$e^+e^- \to B^0\overline{B^0}$ events, low background.

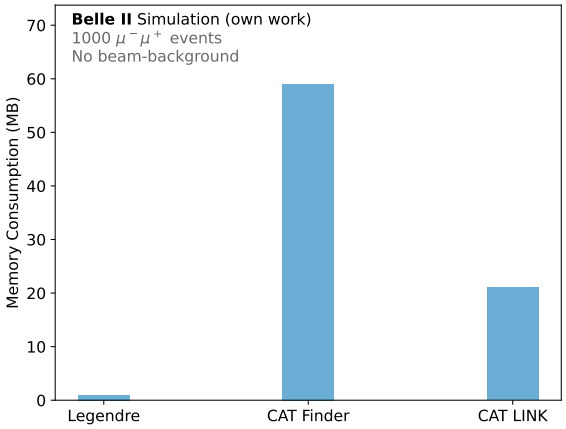


$e^+e^- \to B^+B^-$ events, high background.

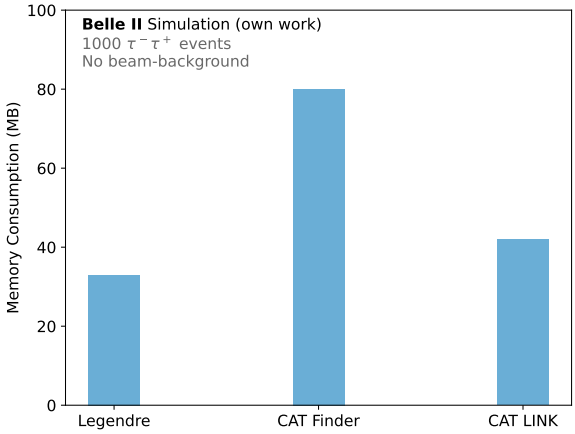


$e^+e^- \to B^0\overline{B^0}$ events, high background.

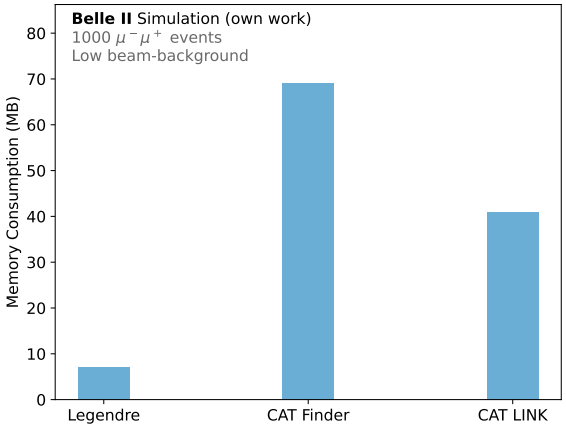
A.2.2 Memory performance



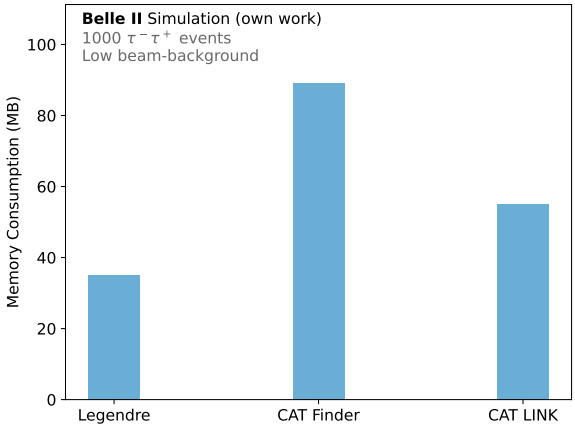
$e^+e^- \to \mu^-\mu^+$ events, no background.



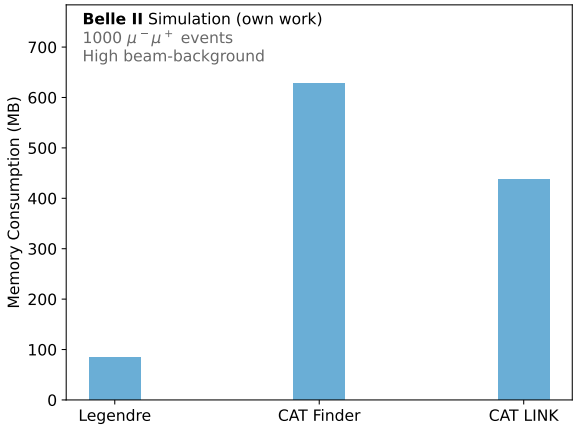
$e^+e^- \to \tau^-\tau^+$ events, no background.



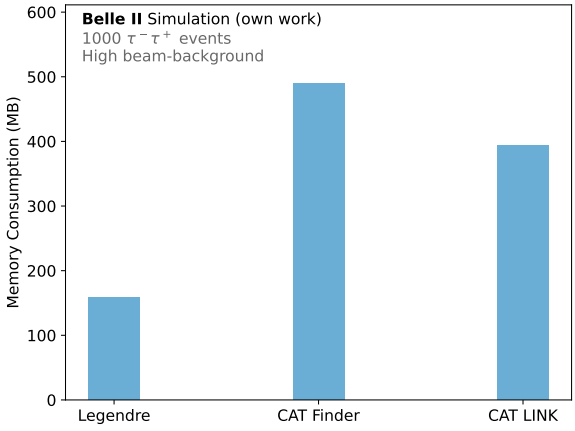
$e^+e^- \to \mu^-\mu^+$ events, low background.



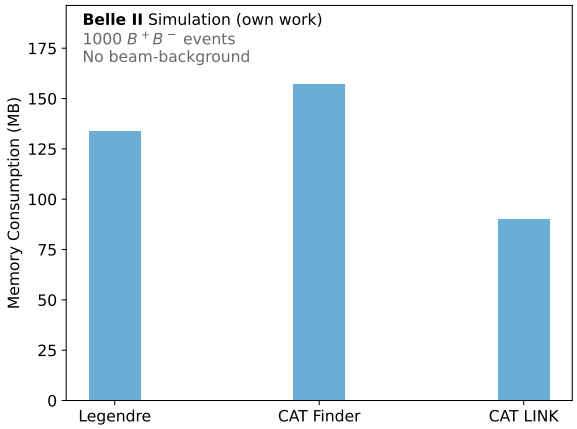
$e^+e^- \to \tau^-\tau^+$ events, low background.



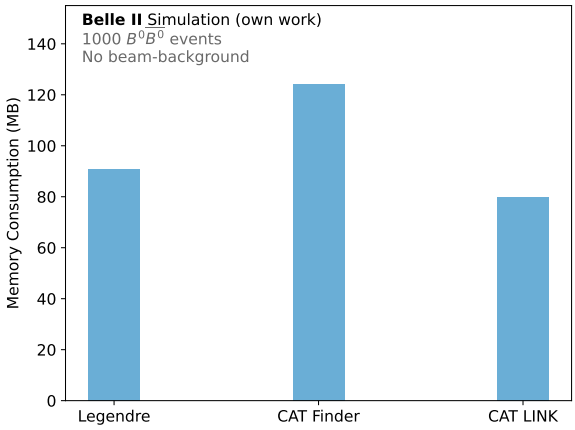
$e^+e^- \to \mu^-\mu^+$ events, high background.



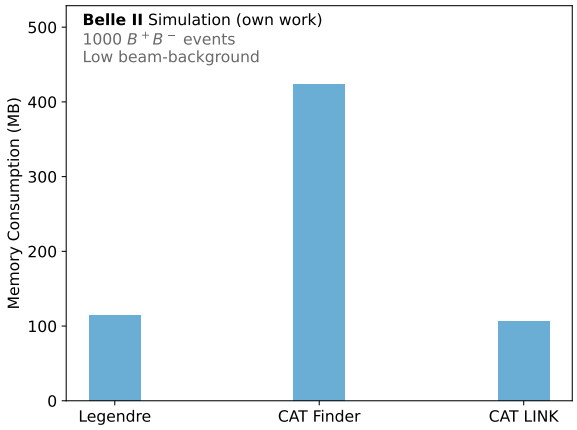
$e^+e^- \to \tau^-\tau^+$ events, high background.



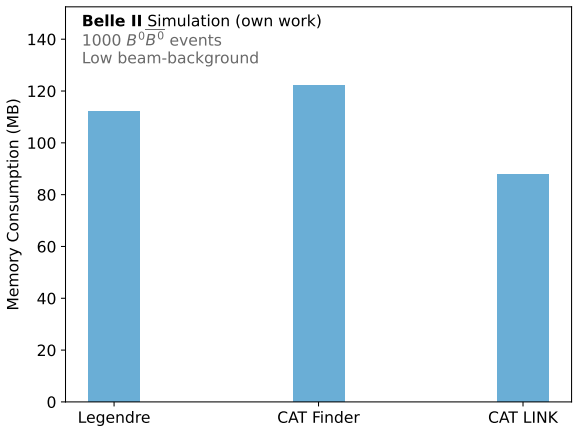
$e^+e^- \to B^+B^-$ events, no background.



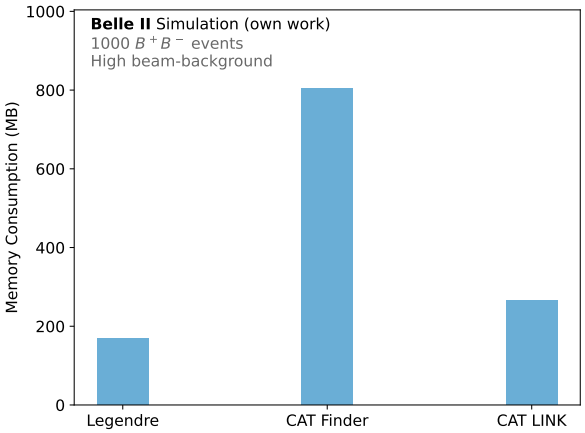
$e^+e^- \to B^0\overline{B^0}$ events, no background.



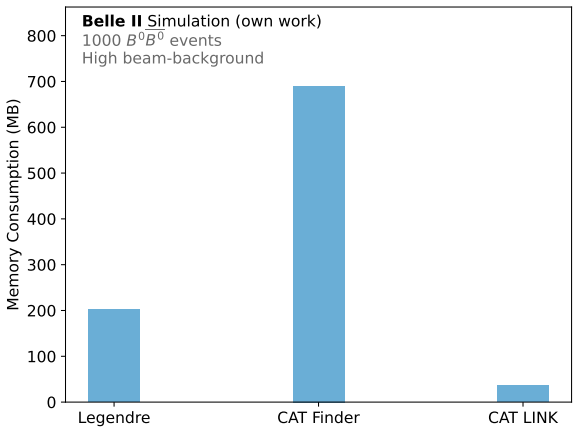
$e^+e^- \to B^+B^-$ events, low background.



$e^+e^- \to B^0\overline{B^0}$ events, low background.

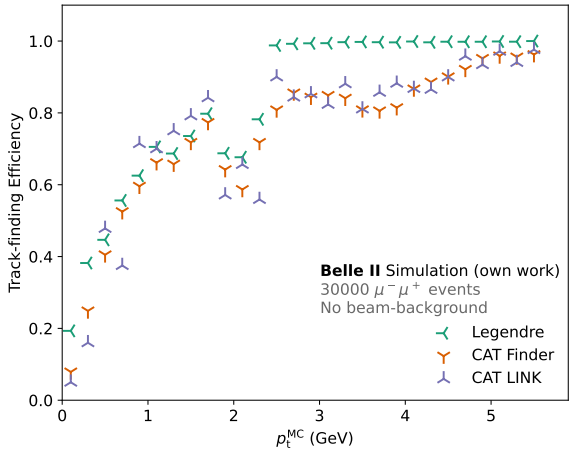


$e^+e^- \to B^+B^-$ events, high background.

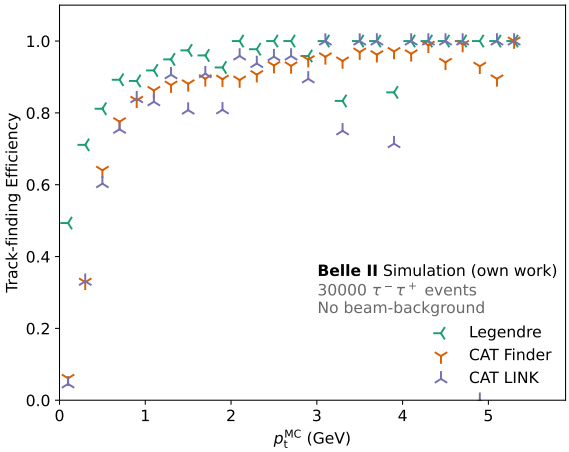


$e^+e^- \to B^0\overline{B^0}$ events, high background.

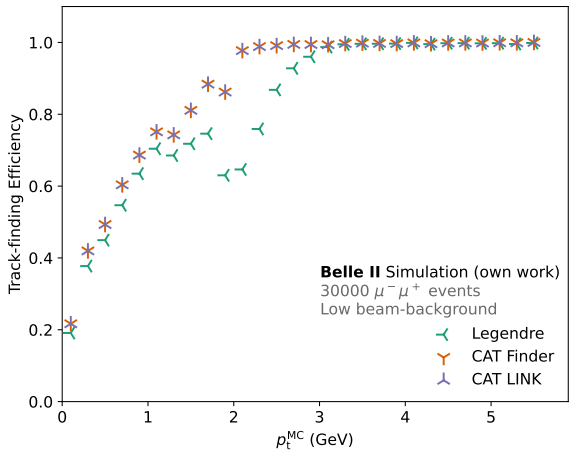
A.2.3 Track finding efficiency



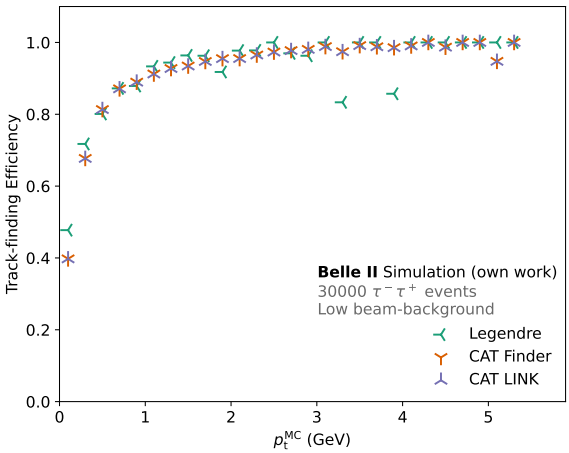
$e^+e^- \to \mu^-\mu^+$ events, no background.



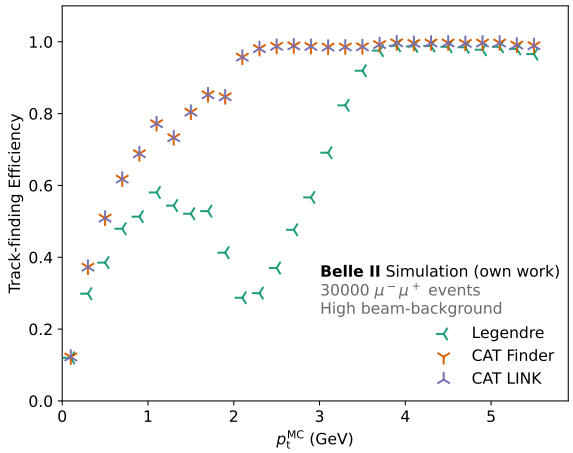
$e^+e^- \to \tau^-\tau^+$ events, no background.



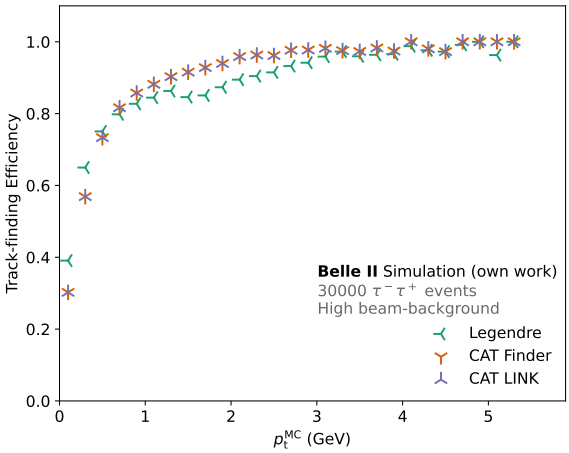
$e^+e^- \to \mu^-\mu^+$ events, low background.



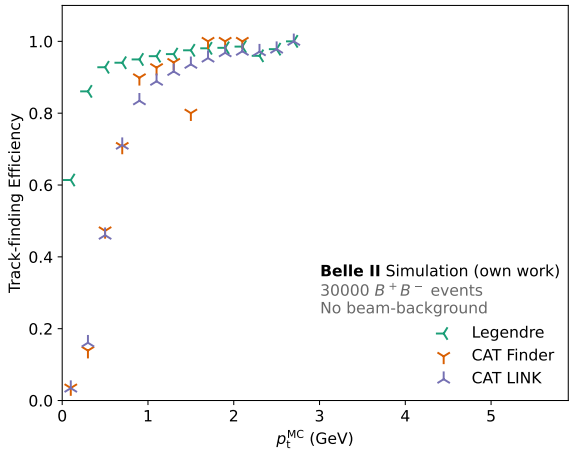
$e^+e^- \to \tau^-\tau^+$ events, low background.



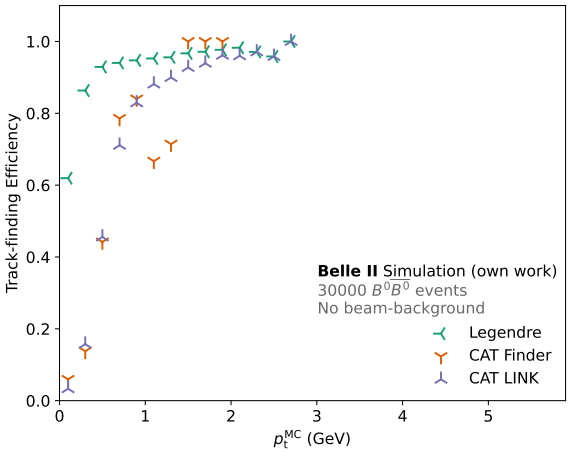
$e^+e^- \to \mu^-\mu^+$ events, high background.



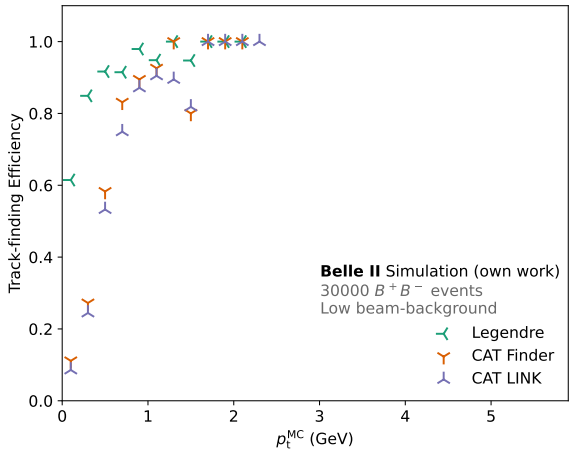
$e^+e^- \to \tau^-\tau^+$ events, high background.



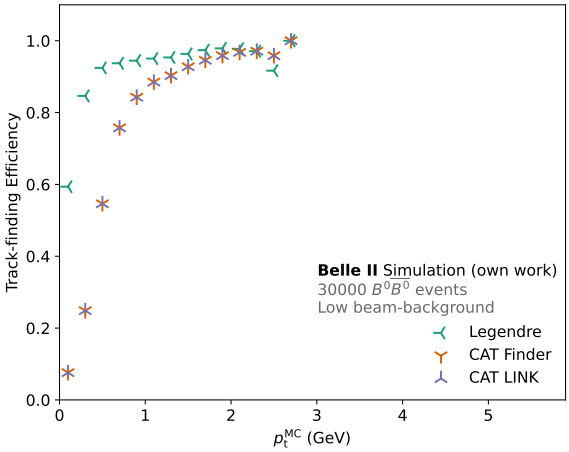
$e^+e^- \to B^+B^-$ events, no background.



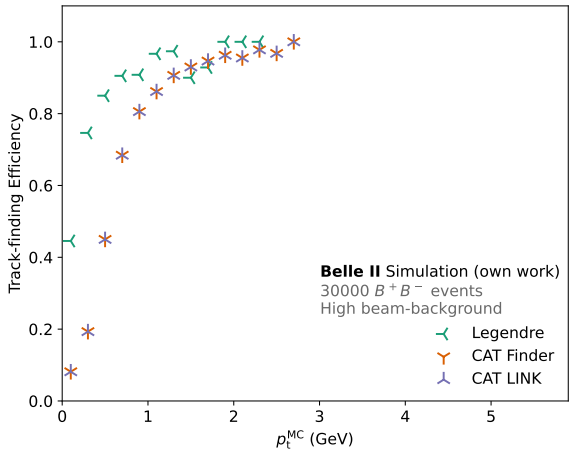
$e^+e^- \to B^0\overline{B^0}$ events, no background.



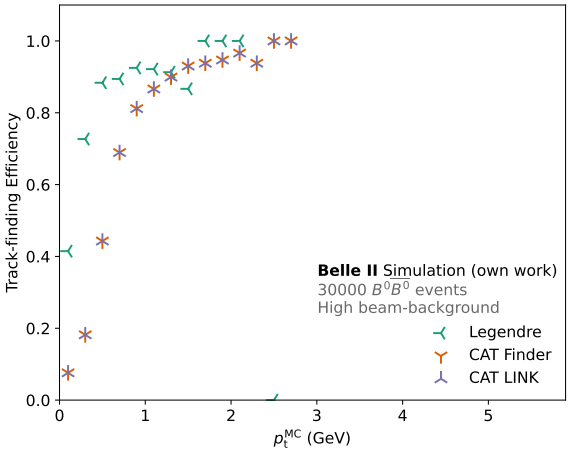
$e^+e^- \to B^+B^-$ events, low background.



$e^+e^- \to B^0\overline{B^0}$ events, low background.

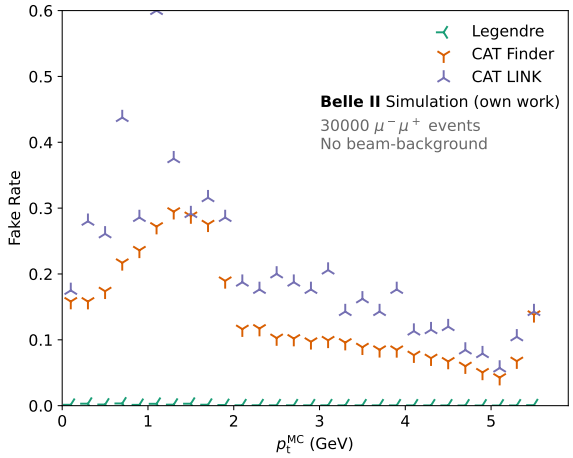


$e^+e^- \to B^+B^-$ events, high background.

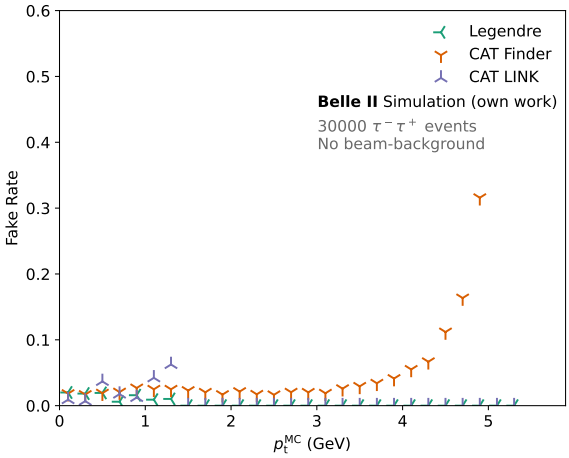


$e^+e^- \to B^0\overline{B^0}$ events, high background.

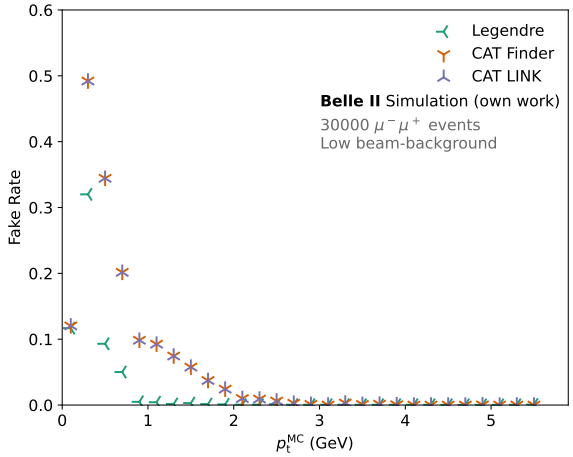
A.2.4 Fake rate



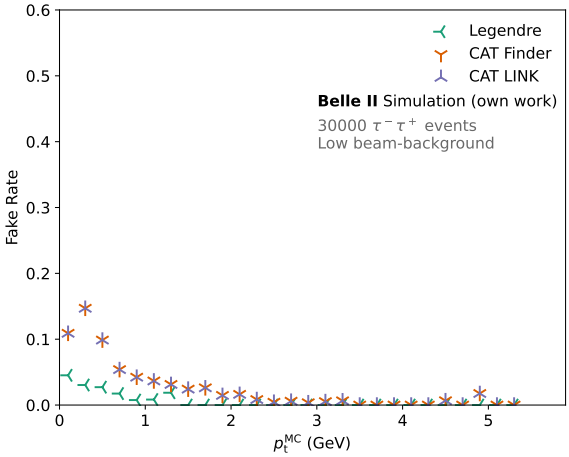
$e^+e^- \to \mu^-\mu^+$ events, no background.



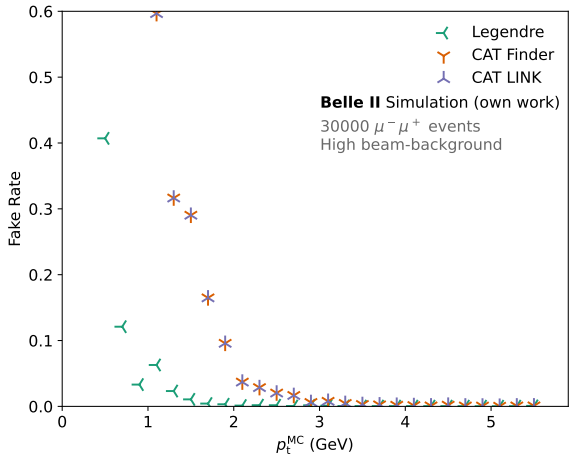
$e^+e^- \to \tau^-\tau^+$ events, no background.



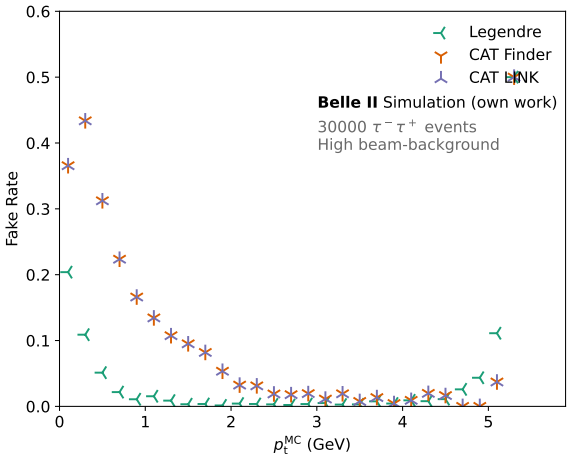
$e^+e^- \to \mu^-\mu^+$ events, low background.



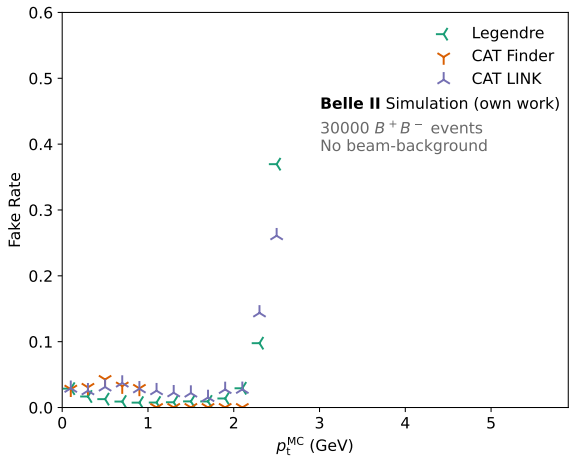
$e^+e^- \to \tau^-\tau^+$ events, low background.



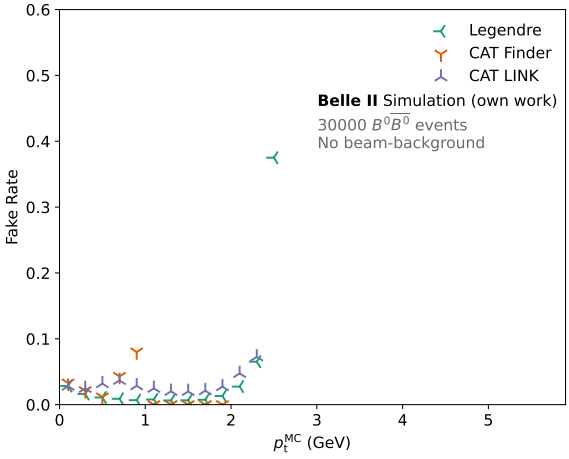
$e^+e^- \to \mu^-\mu^+$ events, high background.



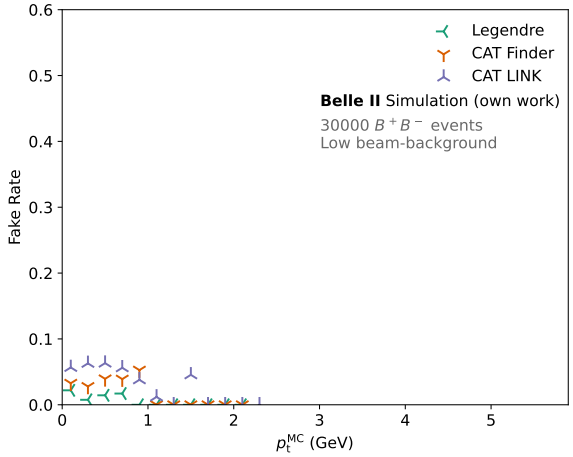
$e^+e^- \to \tau^-\tau^+$ events, high background.



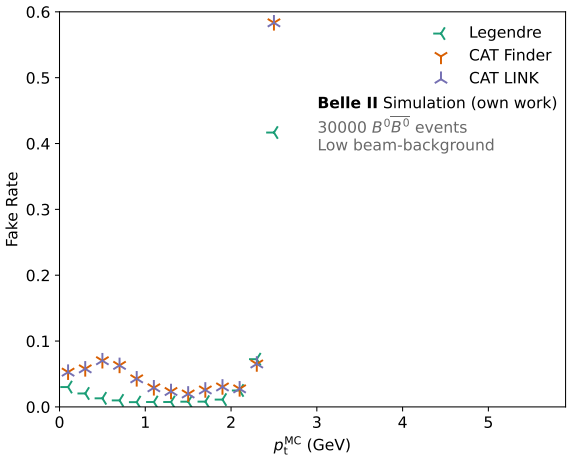
$e^+e^- \to B^+B^-$ events, no background.



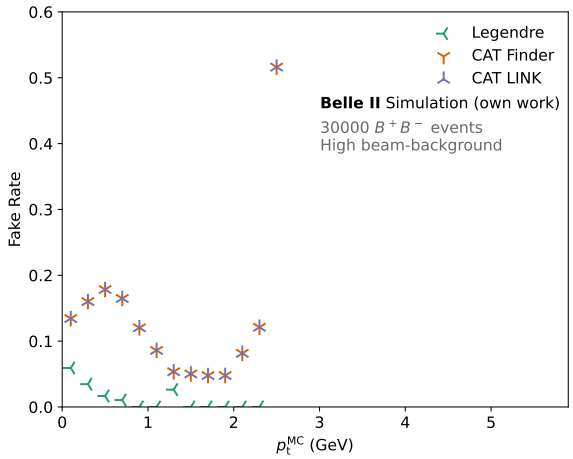
$e^+e^- \to B^0\overline{B^0}$ events, no background.



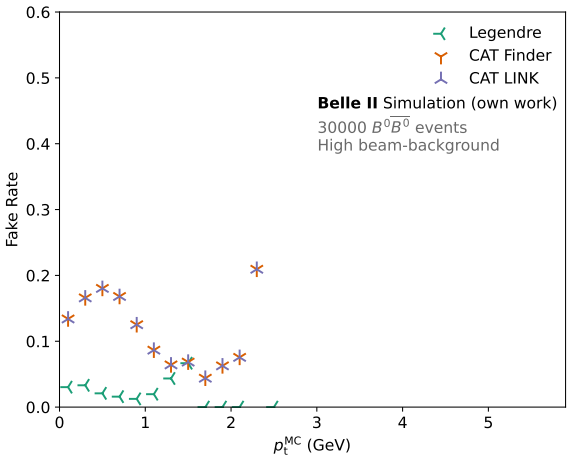
$e^+e^- \to B^+B^-$ events, low background.



$e^+e^- \to B^0\overline{B^0}$ events, low background.

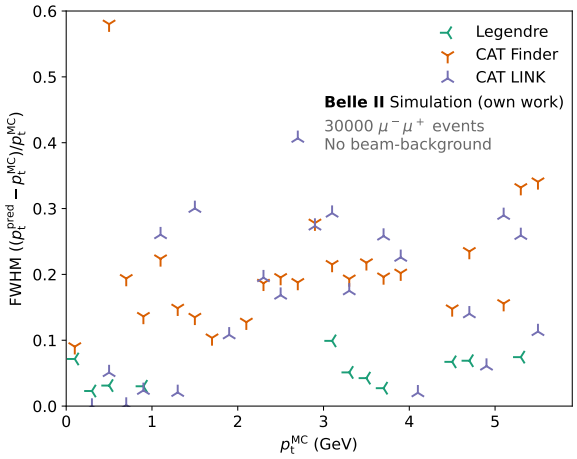


$e^+e^- \to B^+B^-$ events, high background.

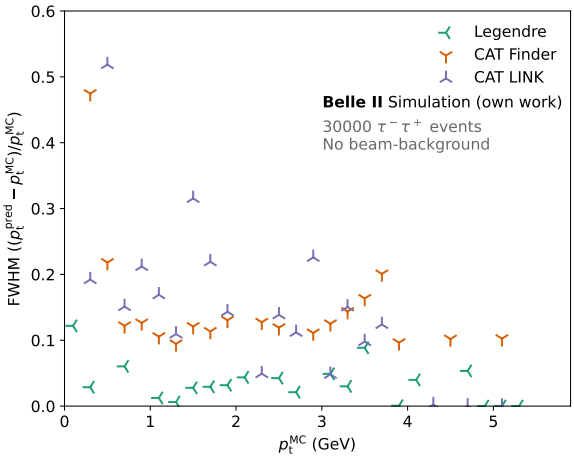


$e^+e^- \to B^0\overline{B^0}$ events, high background.

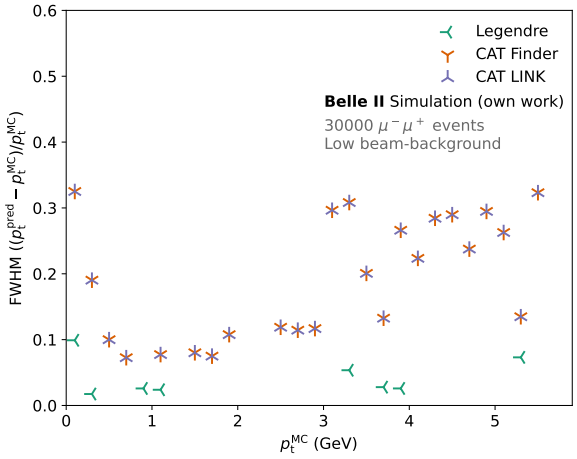
A.2.5 Resolution of transverse momentum



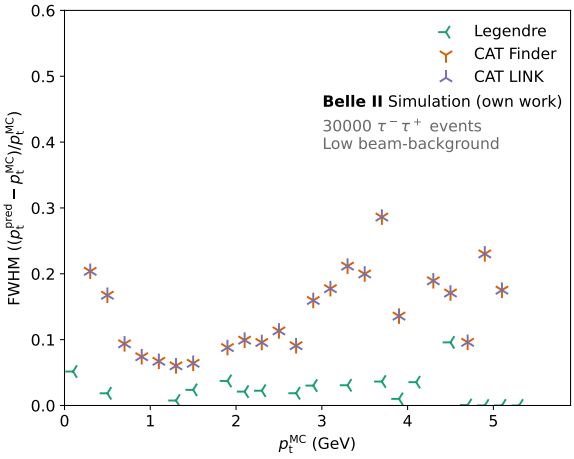
$e^+e^- \to \mu^-\mu^+$ events, no background.



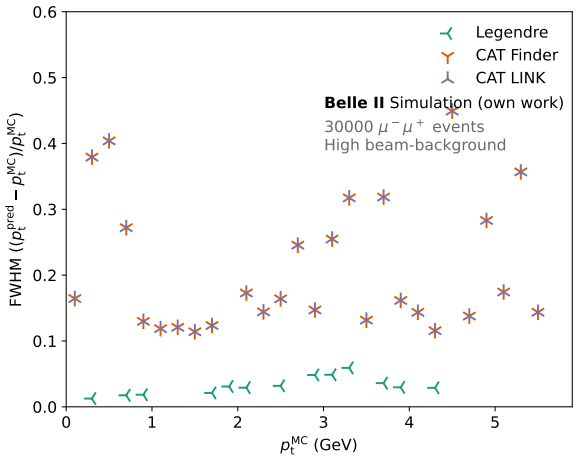
$e^+e^- \to \tau^-\tau^+$ events, no background.



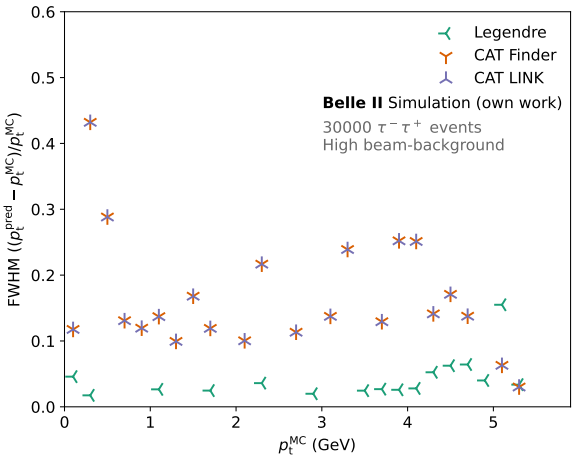
$e^+e^- \to \mu^-\mu^+$ events, low background.



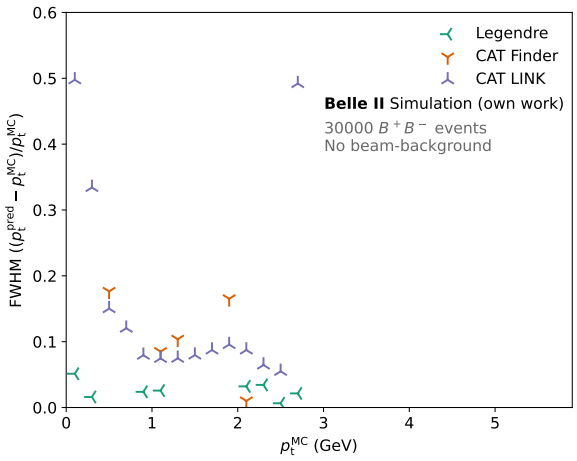
$e^+e^- \to \tau^-\tau^+$ events, low background.



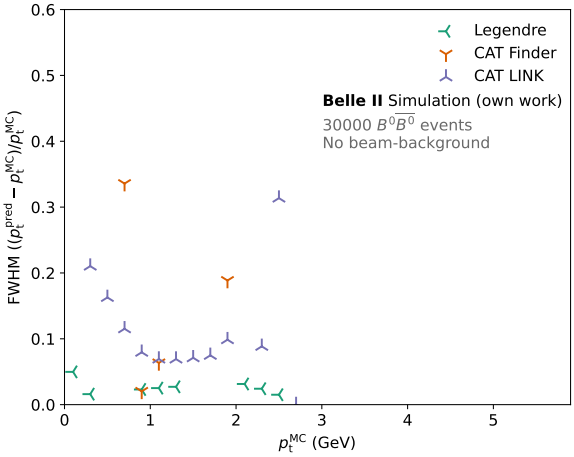
$e^+e^- \to \mu^-\mu^+$ events, high background.



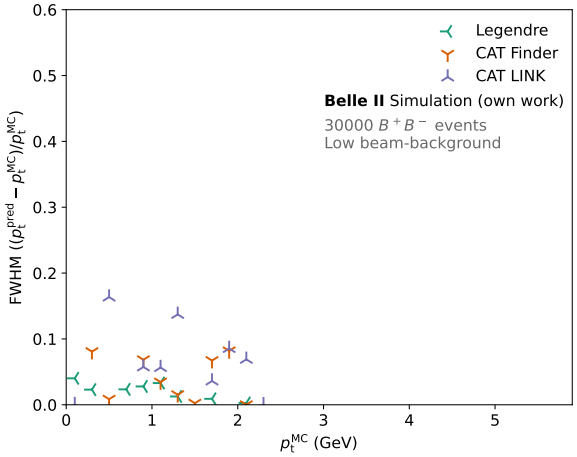
$e^+e^- \to \tau^-\tau^+$ events, high background.



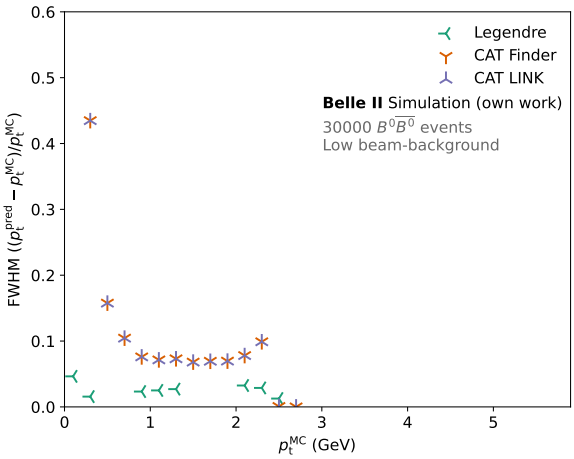
$e^+e^- \to B^+B^-$ events, no background.



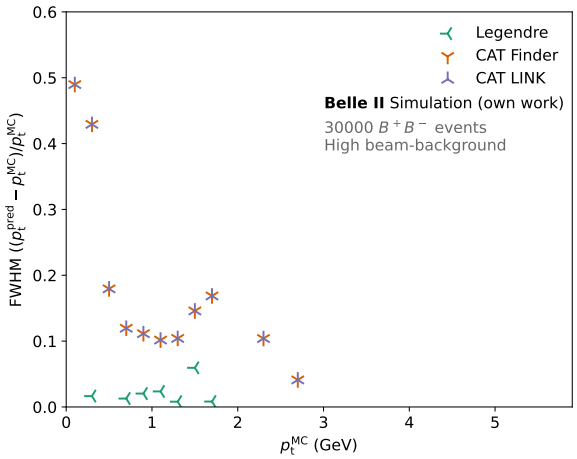
$e^+e^- \to B^0\overline{B^0}$ events, no background.



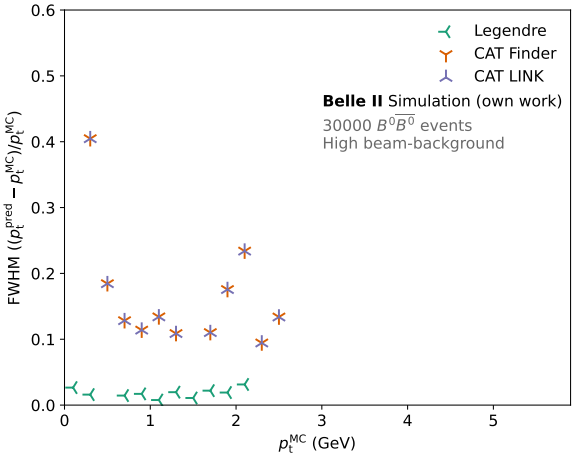
$e^+e^- \to B^+B^-$ events, low background.



$e^+e^- \to B^0\overline{B^0}$ events, low background.



$e^+e^- \to B^+B^-$ events, high background.



$e^+e^- \to B^0\overline{B^0}$ events, high background.

List of Tables

List of Figures

Bibliography

[1] T. Kuhr *et al.*, *The Belle II Core Software*, Computing and Software for Big Science **3** (2019) 1.

[2] L. Reuter, P. Dorwarth, S. Stefkova, and T. Ferber, *Graph Neural Network based Track Finding in the Central Drift Chamber at Belle II*, May, 2023. Contribution to the conference CHEP 2023, https://indico.jlab.org/event/459/.

[3] A. Natochii *et al.*, *Beam background expectations for Belle II at SuperKEKB*, Aug., 2022. arXiv:2203.05731 [hep-ex].

[4] E. Kou *et al.*, *The Belle II Physics Book*, Progress of Theoretical and Experimental Physics **2019** (2019) 123C01.

[5] V. Bertacchi *et al.*, *Track finding at Belle II*, Computer Physics Communications **259** (2021) 107610.

[6] F. Forti, *Snowmass Whitepaper: The Belle II Detector Upgrade Program*, Mar., 2022. arXiv:2203.11349 [hep-ex].

[7] Y. Ohnishi *et al.*, *Accelerator design at SuperKEKB*, Progress of Theoretical and Experimental Physics **2013** (2013) 3A011.

[8] Particle Data Group *et al.*, *Review of Particle Physics*, Progress of Theoretical and Experimental Physics **2022** (2022) 083C01.

[9] T. Abe *et al.*, *Belle II Technical Design Report*, Nov., 2010. arXiv:1011.0352 [hep-ex].

[10] Jojosito *et al.*, *GenFit/GenFit: release-02-00-05*, Dec., 2023. doi: 10.5281/ZENODO.10301439.

[11] M. Duerr *et al.*, *Long-lived Dark Higgs and Inelastic Dark Matter at Belle II*, Journal of High Energy Physics **2021** (2021) 146, arXiv:2012.08595 [hep-ph].

[12] S. R. Qasim, J. Kieseler, Y. Iiyama, and M. Pierini, *Learning representations of irregular particle-detector geometry with distance-weighted graph networks*, The European Physical Journal C **79** (2019) 608, arXiv:1902.07987 [hep-ex, stat].

[13] J. Kieseler, *Object condensation: one-stage grid-free multi-object reconstruction in physics detectors, graph and image data*, The European Physical Journal C **80** (2020) 886, arXiv:2002.03605 [hep-ex].

[14] L. Reuter, *Graph Neural Network based Track finding in the CDC*, Contribution to the Belle II Germany Meeting 2023, https://indico.belle2.org/event/8188/.

[15] A. Paszke *et al.*, *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, Dec., 2019. arXiv:1912.01703 [cs, stat].

[16] The Belle II Collaboration, *Belle II Analysis Software Framework (basf2)*, Aug., 2022. doi: 10.5281/ZENODO.5574115.

[17] *List of Core Modules — basf2 light-2311-nebelung documentation*, https://training.belle2.org/framework/doc/index-04-modules.html, last accessed March 24th 2024.

[18] S. Agostinelli *et al.*, *Geant4—a simulation toolkit*, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **506** (2003) 250.

[19] D. J. Lange, *The EvtGen particle decay simulation package*, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **462** (2001) 152.

[20] *belle2 externals*, Oct., 2022. original-date: 2021-07-29T11:50:38Z, https://github.com/belle2/externals.

[21] R. Brun *et al.*, *root-project/root: v6.18/02*, Aug., 2019. doi: 10.5281/ZENODO.3895860.

[22] *Python/C API Reference Manual*, https://docs.python.org/3/c-api/index.html, last accessed March 1st 2024.

[23] S. Jadach, B. F. L. Ward, and Z. Was, *The precision Monte Carlo event generator for two-fermion final states in collisions*, Computer Physics Communications **130** (2000) 260.

[24] S. Jadach, J. H. Kühn, and Z. Was, *TAUOLA - a library of Monte Carlo programs to simulate decays of polarized leptons*, Computer Physics Communications **64** (1991) 275.

[25] *argparse — Python documentation*, https://docs.python.org/3/library/argparse.html, last accessed March 25th 2024.

[26] *time — Python documentation*, https://docs.python.org/3/library/time.html, last accessed March 1st 2024.

[27] T. Skwarnicki, *A study of the radiative CASCADE transitions between the Upsilon-Prime and Upsilon resonances*, tech. rep., [object Object], 1986. Artwork Size: pages 133 Publication Title: 133 pp. (1986)., doi: 10.3204/PUBDB-2023-03027.

[28] *SciPy v1.12.0 Manual*, https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.crystalball.html, last accessed February 27th 2024.

[29] F. Wemmer *et al.*, *Photon Reconstruction in the Belle II Calorimeter Using Graph Neural Networks*, Computing and Software for Big Science **7** (2023) 13.

[30] M. Fey and J. E. Lenssen, *Fast Graph Representation Learning with PyTorch Geometric*, Apr., 2019. arXiv:1903.02428 [cs, stat].

[31] Y. Klügl, *CAT LINK*, Mar., 2024. doi: 10.5281/ZENODO.10895311.

Declaration of Authorship

I declare that the enclosed thesis has been composed by me and is based on my own work unless stated otherwise. No other person's work has been used without due acknowledgment in this thesis.
Additionally, I acknowledge the KIT statutes for safeguarding good research practice in the September 30th, 2021 version.

Karlsruhe, March 30th 2024

.....
(Yannis Klügl)